Leon3 NoC System Generator

ROYAL INSTITUTE
OF TECHNOLOGY

A thesis submitted in partial fulfillment for the
Master Degree in System on Chip Design

By: Jawwad Raza Syed

KTH Royal Institute of Technology
ICT/Electronics, September 2010

Professor: Dr. Johnny Öberg
Examiner: Dr. Ingo Sander

Abstract

In order to meet challenges of today's computation extensive SoC applications, there is a need to have more processors on a chip than ever before. Network on Chip (NoC) is a new paradigm to meet these challenges. It provides high computation power by incorporating multiple processors on a single integrated circuit (IC) in the form of a network. The NoC provides high bandwidth and high computation power and avoids the problems faced by using conventional bus systems. The NoC designs are developed so far by using design tools and platform of different venders. Each has their strict license policies and development limitations. With the advent of the NoC, there arises a need to make NoC based systems more efficient as well as low cost. To achieve this goal, the NoC systems must be made technology independent, by using the freely available opencores and opensource processors and IP cores. The Leon3 processor developed by AeroFlex Gaisler is a good choice for this purpose. The Leon3 uses the AMBA bus to connect to different peripherals and IP cores that are also provided by AeroFlex Gaisler in the form of a library. The thesis is targeted to migrate from the existing multi-core 4x4 NoC system that is based on Nios II processor and developed by KTH to the 2D Mesh 2x2 NoC system that is based on Leon3 processor. There is also provided a Graphical User Interface (GUI) to generate the Leon3 NoC system.

Acknowledgement

I am very thankful to my supervisor Dr. Johnny Öberg and my examiner Dr. Ingo Sander for their continuous support and their patience with me and for providing me the opportunity to work on this emerging technology.

I am also thankful to Jiri Gaisler and his team for providing me support throughout my thesis work.

Contents

Abstract iii
Acknowledgement v
Figures xi
Tables xii
Abbreviations xiii
Chapter 1 1
1. Introduction 1
1.1. Background 1
1.1.1. Network on Chip 1
1.1.2. The Thesis 2
1.2. Outline 2
1.2.1. Chapter 2: Background 2
1.2.2. Chapter 3: The NoC 2
1.2.3. Chapter 4: Design Setup 2
1.2.4. Chapter 5: Methodology 3
1.2.5. Chapter 6: Evaluation 3
1.2.6. Chapter 7: Conclusion and Future work 3
Chapter 2 5
2. Background 5
2.1. AMBA Bus Architecture 5
2.1.1. Advanced High-performance Bus (AHB) 5
2.1.2. Advanced System Bus (ASB) 5
2.1.3. Advanced Peripheral Bus (APB) 6
2.2. AMBA AHB components 6
2.2.1. AMBA AHB operation 6
2.2.2. AHB Bus Transfer 7
2.2.3. Data buses 8
2.2.4. AHB transfer direction 8
2.2.5. AMBA AHB Signals 8
2.2.6. Address decoding 9
2.2.7. AHB bus slave 9
2.3. The Leon3 Processor 10
2.3.1. The Integer unit 11

2.3.2. RAM usage 12
2.4. The GRLIB IP library 13
2.4.1. Available IP cores 13
2.4.2. Library Organization 14
2.4.3. Design Concept 14
2.4.4. On-chip Bus interconnection 14
2.4.5. AHB Slave Interface 15
2.4.6. AHB bus Index Control 16
2.4.7. The Plug&Play capability 16
2.4.8. Portability 16
2.4.9. AHBRAM - Single-port RAM with AHB interface 17
2.4.10. AHBROM - Single-port ROM with AHB interface 17
2.4.11. SYNCRAM_DP - Dual-port RAM generator 17
2.5. Wrapper 17
2.5.1. Overview 17
2.6. The Hardware 18
2.7. GUI Development Environment 19
2.7.1. Visual Basic 2008 19
2.7.2. The Graphical User Interface 19
2.7.3. The .NET Framework 19
2.7.4. VB Shell Function 20
Chapter 3 21
3. Network on Chip 21
3.1. Basic NoC 21
3.1.1. Resource 22
3.1.2. Resource Network Interface (RNI) 22
3.1.3. Links 22
3.1.4. Switch 22
3.1.5. Network Topology 22
3.1.6. Flow Control 23
3.1.7. Routing Algorithm 24
3.2. 2D Mesh 2x2 NoC 24
3.2.1. Overview 24
3.2.2. Switch architecture 25
3.2.3. Node Address Decoding 25
3.2.4. The Possible Routing directions 26

3.2.5. The Packet Format 26
Chapter 4 29
4. Design Setup 29
4.1. Requirements 29
4.2. GRLIB Installation 29
4.2.1. Directory Organization 29
4.2.2. Host platform support 30
4.3. GRTools 30
4.3.1. Windows with Cygwin 30
4.4. The Working of GRTools 31
4.5. Implementation 31
4.6. AHB plug&play configuration 31
4.6.1. Device identification 32
4.7. Leon3 Configuration 33
4.7.1. Synthesis 33
4.7.2. Clock Generation 33
4.7.3. Processor 34
4.7.4. Integer Unit 34
4.7.5. AMBA Configuration 35
4.7.6. Peripherals 36
4.7.7. On-chip RAM/ROM 36
4.8. Simulation 37
4.9. Synthesis and place&route 37
4.9.1. Running applications on target 37
Chapter 5 41
5. Methodology 41
5.1. Resource Network Interface 41
5.1.1. Communication between RNI and Switch 42
5.1.2. Routing Management 42
5.2. The wrapper 42
5.2.1. Wrapper Architecture 43
5.3. RNI adaption in the design 44
5.3.1. RNI memory 45
5.4. AHBROM modification 45
5.5. Top Design File 46
5.5.1. Design Synthesis 48

5.6. Graphical User Interface 49
5.6.1. Overview 49
5.6.2. Generation of a new system 50
5.6.3. Open an existing system 51
5.6.4. Configuring Nodes 51
5.6.5. Simulation 52
5.6.6. Synthesis 52
Chapter 6 55
6. Evaluation 55
6.1. Adaption of RNI memory in leon3mp project 55
6.2. Modification of AHBROM 55
6.2.1. Generating Boot Image 56
6.3. Testbench implementation 56
6.3.1. Structure 56
6.3.2. Working 56
6.4. Synthesis using Quartus 57
6.5. Design compilation 59
6.6. Design porting on FPGA board 61
6.6.1. Accessing Hardware 61
6.7. Changes required in GRLIB 62
Chapter 7 63
7. Conclusion & Future work 63
7.1. Conclusion 63
7.2. Future work 63
References 65

Figures

Figure 1: AMBA Bus system topology 5
Figure 2: Simple bus transfer 7
Figure 3: AHB transfer with wait states 8
Figure 4: AHB bus slave interface 10
Figure 5: Leon3 processor architecture 11
Figure 6: Leon3 configuration register 12
Figure 7: AHB Interconnection view 14
Figure 8: The wrapper concept 18
Figure 9: Nios II development board Stratix II edition 19
Figure 10: The 9 nodes Mesh NoC 21
Figure 11: Node representation 22
Figure 12: 2x2 NoC pattern 25
Figure 13: 2x2 NoC node IDs 26
Figure 14: Possible Packet routing 26
Figure 15: 2x2 NoC Packet format 27
Figure 16: The plug&play configuration 32
Figure 17: Leon3 Design Configuration 33
Figure 18: Synthesis Menu 33
Figure 19: Clock generation 34
Figure 20: Processor Menu 34
Figure 21: Integer unit menu 35
Figure 22: AMBA Configuration menu 35
Figure 23: Peripherals menu 36
Figure 24: On-chip RAM/ROM peripheral menu 36
Figure 25: RNI memory configuration 41
Figure 26: The NoC Wrapper 43
Figure 27: Multiple Leon3 processors on single AMBA bus 46
Figure 28: The Top module 47
Figure 29: 2x2 NoC Design 47
Figure 30: RTL view of the 2x2 Leon3 based NoC system 49
Figure 31: GUI for 2x2 NoC system 50
Figure 32: Folder browser for generating new system 50
Figure 33: Opens an existing system 51
Figure 34: Node configurations 51
Figure 35: Modelsim simulator 52
Figure 36: Quartus II design environment 53
Figure 37: The testbench simulation 57
Figure 38: RNI evaluated in Leon3mp design 58
Figure 39: The normal execution flow graph 59
Figure 40: the flow graph with .mif converter 60
Figure 41: Node configuration and compilation 61

Tables

Table 1: AMBA AHB signals 9
Table 2: syncram_2p sizes for Leon3 register file 13
Table 3: GRLIB folders 29
Table 4: GRLIB directory organization 30
Table 5: RNI wrapper signals 43
Table 6: Compilation report summary 57
Table 7: RAM Summary 59

Abbreviations

AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric Multiprocessing
APB	Advanced Peripheral Bus
ARM	Advanced RISC Machines
ASB	Advanced System Bus
ASIC	Applications Specific Integrated Circuit
ASR	Application Specific Register
CAN	Controller Area Network
DDR	Double Data Rate
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
GNU	GNU Compiler Collection
GPIO	General Purpose Input/Output
GPL	General Public License
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
LRR	Least Recently Replaced
LUT	Look Up Table
LRU	Least Recently Used
MAC	Multiply Accumulate
MMU	Memory Management Unit
NoC	Network on Chip
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
ROM	Read Only Memory
SDRAM	Synchronous dynamic random access memory
SMP	Symmetric Multiprocessing
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TLB	Translation Lookaside Buffer
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VB	Visual Basic
VHDL	VHSIC (Very High Speed Integrated Circuits) Hardware Description Language

Chapter 1

1. Introduction

This chapter will provide some background about the NoC based systems and the need for generating a Leon3 based NoC system. The chapter will also provide the outline of the later chapter of the thesis.

1.1. Background

With the advent of emerging technologies particularly in the field of embedded systems, the total number of transistors that can be fabricated on an IC will continue to rise and it is estimated that it will grow over one billion in the next decade or so. The IC designers are facing the major challenges to get the maximum performance while keeping the cost of the design under control [1]. It is now becoming the basic requirement that the designer must produce the functionally correct and the reliable systems while keeping the cost low. There are several factors that limit the performance of these systems and causes increase in the energy consumption, among them is the on-chip physical interconnections among components. These connections become more critical when there is requirement of the high bandwidth and high throughput [5]. The high bandwidth requirement forces the designers to increase the size of the design and this raises the design cost.

1.1.1. Network on Chip

Network on Chip (NoC) is a new model for designing large SoC designs. Conventional SoC design faces number of design problems and now there is requirement of a breakthrough. With NoC effort is made to resolve the problems related to the future systems on chip (SoC) for their design productivity, usability and architectures [10].

In NoC design the resource share common physical links in parallel mechanism, therefore the overall data throughput is high and concurrent transactions are possible [9]. The communication takes place by using uniform network that connects the on-chip resources with each other; in this way better bandwidth, scalability and reduced wire delay can be achieved, than that of conventional bus architectures. However there are some overheads of the NoC system that are related to the area and performance, comparing to the optimized dedicated hardware solutions [10].

Another important feature of the NoC design is that it consumes less power. The reasons for this low power consumption is that the links between the nodes are segmented, shortened and distributed evenly. The data in the network has more than one path to travel and this reduces the travel time, switching and power consumption. The NoC can also be utilized as an IP core together with the other IPs that are previously developed by different venders. The NoC IPs reusability enables the designer to design complex SoC with very short design cycles. Their advantage is that the designs are optimized and tested before thus save lot of time and effort. This make it easy to meet the time to market requirements. That is why the NoC is a

promising solution to cope with the limitations of the present communication infrastructure [9].

1.1.2. The Thesis

The thesis is about the design implementation of a Leon3 based 2D mesh 2x2 NoC system. The current design is a Leon3-ported version of a Quadcore 4x4 NoC design developed by KTH [12]. The Quadcore NoC is based on Altera's Nios II processor which uses the Avalon switch fabric bus architecture for communicating with its peripherals. Contrary to the Nios II processor as in the previous case, in the current thesis the Leon3 processor is used as a resource,. The resource (Leon3) is connected to the Switch through an interface called the resource network interface (RNI). The Leon3 is a softcore processor, developed by AeroFlex Gaisler; it is based on AMBA bus architecture. The core for AMBA bus and Leon3 processor and all the related peripherals like SRAM and PROM are available in GRLIB IP library, this IP library is also developed by AeroFlex Gaisler [4].

The GRLIB IP library supports up to four Leon3 processors on an AMBA bus. The GRLIB IP library supports only one bus system at a time, whereas for developing the 2x2 NoC there is requirement to have the multiple bus system in the design project. To make use of this library for the multiple bus system, a top design module has been developed that allows four Leon3 processor based systems, to interact with each other and forming a 2x2 NoC design. The RNI is added in the system as AHB slave, i.e., the slave memory. The information transferred between the switches in the form of packets. The packet contains both data and header fields. The header contains the routing information. A Graphical User Interface is also developed to provide a user friendly environment for generating a Leon3 based 2x2 NoC system. The Graphical User Interface (GUI) is developed to facilitate the users by providing a single platform to generate and configure the new system. The GUI is developed in Microsoft Visual Basic 2008.

1.2. Outline

1.2.1. Chapter 2: Background

This chapter will explain the basic concept necessary to consign the context of the thesis. It provides the description about the Leon3 processor, the AMBA bus architecture particularly the AMBA AHB bus system, the GRLIB IP library, and other related modules.

1.2.2. Chapter 3: The NoC

This chapter will present an overview of the NoC, the basic terminologies and concept about the NoC system along with the details about current NoC system. There will be some explanation about the target hardware, and details about the address decoding and packet formatting.

1.2.3. Chapter 4: Design Setup

This chapter will explain the working of the standalone Leon3 system and the Leon3 configuration procedure. Also the testing of a standalone system using GRLIB IP library and GRTools will be discussed.

1.2.4. Chapter 5: Methodology

This chapter will explain the method and the steps followed to add the RNI wrapper and the adaption of RNI as a slave memory in the Leon3 design. The modification of AHBROM for the adaption of memory initialization file, the development of the Top module and the development of GUI will also be discussed.

1.2.5. Chapter 6: Evaluation

This chapter will explain the development of the testbench for testing the 2x2 NoC design, the compilation and synthesis of the complete system and the process for porting the design on the hardware.

1.2.6. Chapter 7: Conclusion and Future work

This chapter will sum up the objectives achieved and summaries the results obtained. The chapter also highlights the future possibilities that can be carried out.

Chapter 2

2. Background

This chapter highlights some basic concepts that are required to understand, before reading the later chapters. The chapter explains the AMBA bus architecture, the Leon3 processor architecture and some details about the GRLIB IP library and other related modules.

2.1. AMBA Bus Architecture

The AMBA stands for The Advanced Microcontroller Bus Architecture. It is a bus standard developed by ARM ®. The AMBA specification can be regarded as an on-chip communications standard for designing high performance embedded microcontrollers. The typical AMBA bus system is shown in the figure below, here there are two bus systems, one requiring high performance for the high speed components, like, the on-chip memory and DMA are connected to the high performance bus, whereas the other that do not need such high bandwidth are connected through a bridge to the low power bus [2].

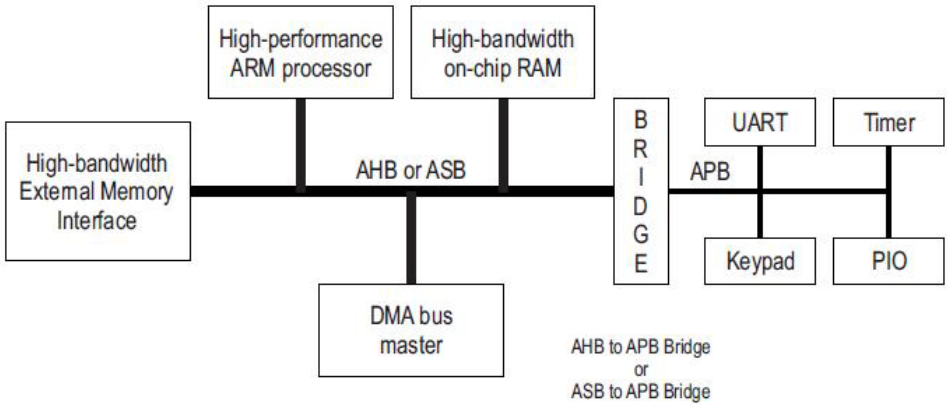


Figure 1: AMBA Bus system topology

Three distinct buses are defined within the AMBA specification:

2.1.1. Advanced High-performance Bus (AHB)

The AMBA AHB is the high-performance system backbone bus. It is for the high performance, high clock frequency system modules. It supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macro-cell functions. AHB is also specified to ensure ease of use in an efficient design flow by using synthesis and automated test techniques [2].

2.1.2. Advanced System Bus (ASB)

AMBA ASB is an alternative system bus suitable for use where the high performance features of AHB are not required. ASB also supports the efficient connection of processors, on-chip

memories and off-chip external memory interfaces with low-power peripheral macro-cell functions [2].

2.1.3. Advanced Peripheral Bus (APB)

AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. The APB is for the low power peripherals. APB can be used in conjunction with either version of the system bus i.e., AHB or ASB [2].

2.2. AMBA AHB components

The Leon3 processor uses AHB bus to communicate with its high speed modules. The AHB is a new generation of AMBA bus which is intended to address the requirements of high-performance synthesizable designs. AMBA AHB is a new level of bus which sits above the APB and implements the features required for high-performance, high clock frequency systems including burst transfers, split transactions, single cycle bus master handover, single clock edge operation, non tri-state implementation and wider data bus configurations (64/128 bits) [2].

A typical AMBA AHB system design contains the following components:

AHB master

A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at one time [2].

AHB slave

A bus slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer [2].

AHB arbiter

If there is need to implement more than one master, bus arbiter is required. The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as, ‘highest priority’ or ‘fair access’ can be implemented depending on the application requirements [2].

AHB decoder

The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer. A single centralized decoder is required in all AHB implementations [2].

2.2.1. AMBA AHB operation

To start an AMBA AHB bus transfer, the master asserts a request signal to the arbiter. The arbiter then indicates when the master will be granted the use of the bus. A granted bus master will then starts an AMBA AHB transfer by driving the address and control signals. These signals provide information about the address, direction and width of the transfer, as well as an indication that if the transfer forms a part of a burst. A write data bus is used to move data

from the master to a slave, while a read data bus is used to move data from a slave to the master [2].

2.2.2.AHB Bus Transfer

The transfer in AMBA AHB bus consists of an address and control cycle and one or more cycles for the data. The address cannot be extended and therefore all slaves must sample the address during this time. The data, however, can be extended using the HREADY signal. When LOW this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data [2].

Simple AHB transfer

The simplest AHB transfer consists of two distinct sections. First is the address phase, which lasts only a single cycle. Second is the data phase, which may require several cycles. This is achieved using the HREADY signal. In a simple transfer with no wait states the master drives the address and control signals onto the bus after the rising edge of HCLK. The slave samples the address and control information on the next rising edge of the clock. The slave can start to drive the appropriate response and this is sampled by the bus master on the third rising edge of the clock [2].

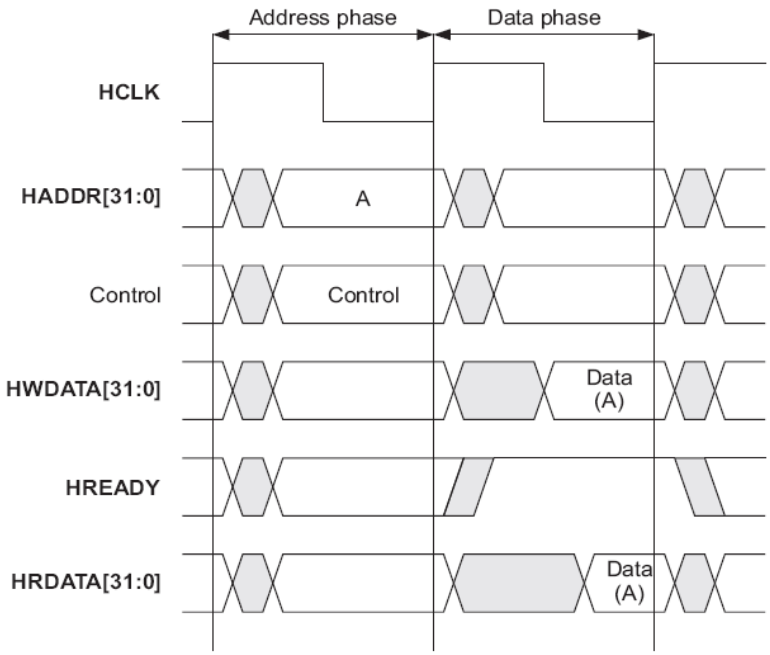


Figure 2: Simple bus transfer

Transfer with wait states

A slave may insert wait states into any transfer, as shown in Figure below, which extends the transfer allowing additional time for completion. For the ‘write operations’ the bus master will hold the data stable throughout the extended cycles. For ‘read operations’ the slave does not have to provide valid data until the transfer is about to complete [2].

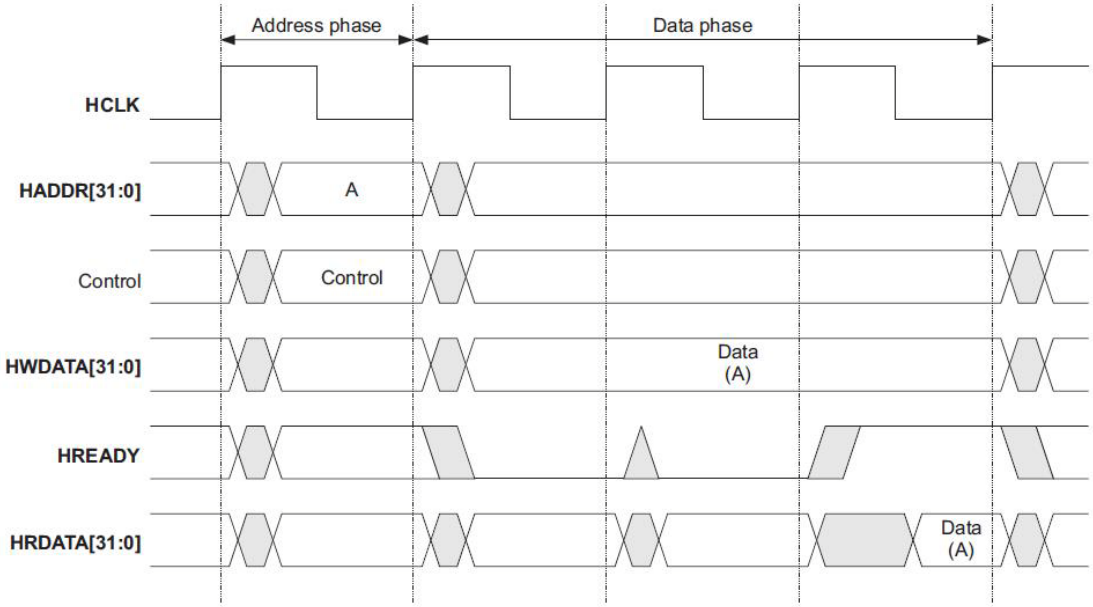


Figure 3: AHB transfer with wait states

2.2.3.Data buses

In AHB system two separate read and write data buses are available. The write data bus HWDATA [31:0] is driven by the bus master during write transfers. If the transfer is extended then the bus master must hold the data valid until the transfer completes, as indicated by HREADY HIGH. The read data bus HRDATA [31:0] is driven by the appropriate slave during read transfers. If the slave extends the read transfer by holding HREADY LOW then the slave only needs to provide valid data at the end of the final cycle of the transfer, as indicated by HREADY HIGH. A slave only has to provide valid data when a transfer completes with an OKAY response. Other responses like SPLIT, RETRY and ERROR do not require valid read data [2].

2.2.4.AHB transfer direction

When HWRITE is HIGH, this signal indicates a write transfer and the master will broadcast data on the write data bus, HWDATA [31:0]. When HWRITE is LOW a read transfer will be performed and the slave must generate the data on the read data bus HRDATA [31:0], for the details about the bus transfer see [2].

2.2.5.AMBA AHB Signals

The AHB signals with their brief description are shown below. The name of all AHB signals are started with the letter H at the beginning [2].

Name	Source	Description
HCLK Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of HCLK.
HRESETn Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0] Address bus	Master	The 32-bit system address bus.
HTRANS[1:0]	Master	Indicates the type of the current transfer, which can be

Transfer type		NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0] Transfer size	Master	Indicates the size of the transfer, which is typically
HBURST[2:0] Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
HPROT[3:0] Protection control	Master	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.
HWDATA[31:0] Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELx Slave select	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
HRDATA[31:0] Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY Transfer done	Slave	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require HREADY as both an input and an output signal.
HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

Table 1: AMBA AHB signals

2.2.6. Address decoding

For each slave on the bus a central address decoder is used to provide a select signal HSELx. A slave must only sample the address and control signals and HSELx when HREADY is HIGH, indicating that the current transfer is completing. Under certain circumstances it is possible that HSELx will be asserted when HREADY is LOW, but the selected slave will have changed by the time the current transfer completes. In the case where a system design does not contain a completely filled memory map an additional default slave should be implemented to provide a response when any of the nonexistent address locations are accessed. Typically the default slave functionality will be implemented as part of the central address decoder [2].

2.2.7. AHB bus slave

If a bus master initiates any transfer within a system then the AHB bus slave responds to that transfer. The slave uses a HSELx select signal from the decoder to determine when it should respond to a bus transfer. All other signals required for the transfer, such as the address and control information, will be generated by the bus master. The AHB bus slave interface is shown below [2].

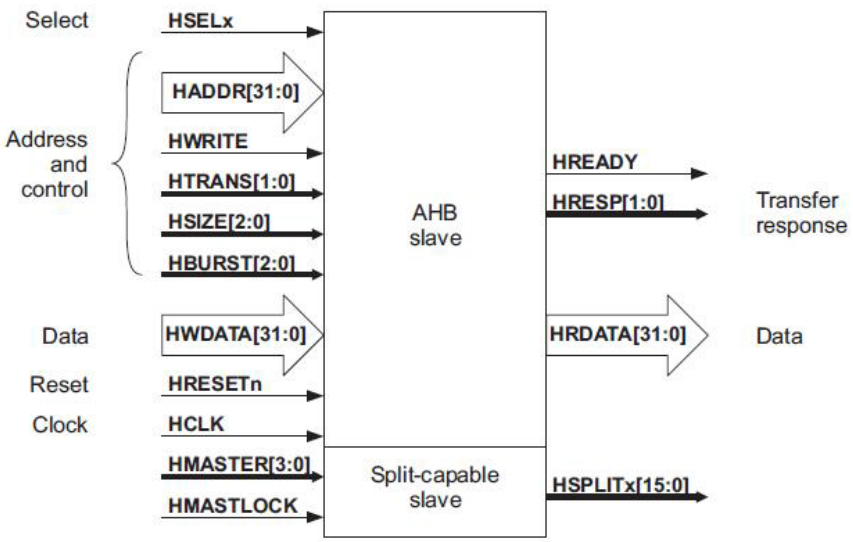


Figure 4: AHB bus slave interface

2.3. The Leon3 Processor

The Leon3 is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The Leon3 model is highly configurable, and very suitable for SoC designs. The full source code is available under the GNU GPL license. The core is available in Gaisler's GRLIB IP library. The Leon3 processor has the following features [17]:

- SPARC V8 instruction set with V8e extensions
- Advanced 7-stage pipeline
- Hardware multiply, divide and MAC units
- High-performance, fully pipelined IEEE-754 FPU
- Separate instruction and data cache (Harvard architecture) with snooping
- Configurable caches: 1 - 4 ways, 1 - 256 Kbytes/way. Random, LRR or LRU replacement
- Local instruction and data scratch pad RAM, 1 - 512 Kbytes
- SPARC Reference MMU (SRMMU) with configurable TLB
- AMBA-2.0 AHB bus interface
- Advanced on-chip debug support with instruction and data trace buffer
- Symmetric Multi-processor support (SMP)
- Power-down mode and clock gating
- Robust and fully synchronous single-edge clock design
- Up to 125 MHz in FPGA and 400 MHz on 0.13 um ASIC technologies
- Fault-tolerant and SEU-proof version available for space applications
- Extensively configurable
- Large range of software tools: compilers, kernels, simulators and debug monitors
- High Performance: 1.4 DMIPS/MHz, 1.8 CoreMark/MHz (gcc -4.1.2)

The Leon3 processor is distributed as part of the GRLIB IP library, allowing the simple integration in the complex SoC designs. In the GRLIB a configurable Leon3 multi-processor design can have up to 4 CPU's and a large range of on-chip peripheral blocks. The Leon3 core has the on-chip debug support and multiprocessor extensions [14].

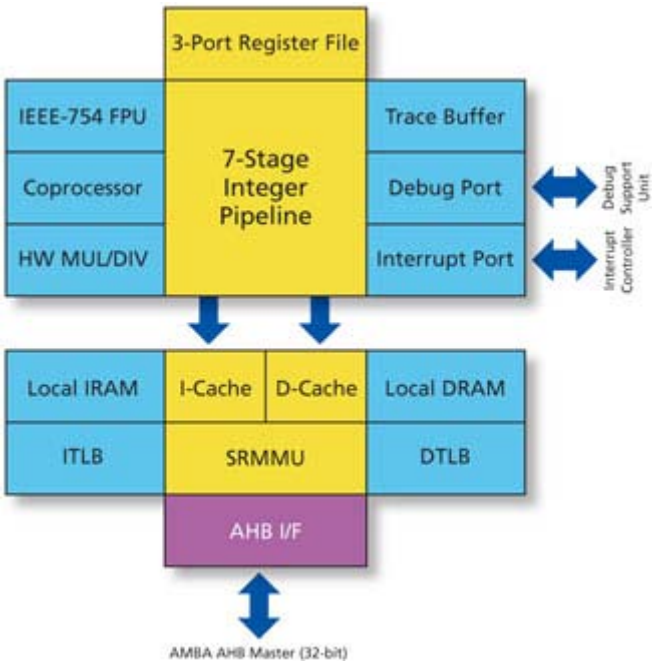


Figure 5: Leon3 processor architecture

The Leon3 is a soft processor, that is, the design is described in a Hardware Description Language (HDL), for example VHSIC Hardware Description Language (VHDL) or Verilog. A design in HDL can be changed before it is synthesized to a netlist. The netlist in its turn can be transferred either to an Application Specific Integrated Circuit (ASIC) design, or implemented on a flexible Field Programmable Gate Array (FPGA) circuit [18].

The Leon3 processor uses a write-through policy, and the cache snooping to maintain cache coherency. The Leon3 processor stores words in the memory in 'Big Endian' order, i.e. the most significant byte is put at the address with the lowest memory value. The Leon3 processor architecture is highly reconfigurable. Parts can be added or taken away and the processor can be configured for a specific application or reconfigured if the conditions for the application change [19].

Following is the brief description of different units of Leon3 processor.

2.3.1. The Integer unit

This is the core unit of Leon3 processor. The integer unit implements the full SPARC V8 standard, including hardware multiply and divides instructions. The number of register windows is configurable within the limit of the SPARC standard (2 - 32). The default setting used is 8. The pipeline consists of 7 stages with a separate instruction and data cache interface, i.e. the Harvard architecture [15].

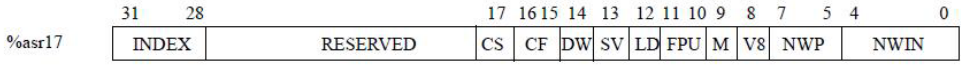
Register windows

The term 'Register windows' is important in the SPARC V8 architecture, it uses a sliding register window to manage how programs access the available registers. At any time a piece of software will have access to four sets of eight registers each. The first set is "global" and can always be accessed. The next three sets make up the register "window". In the "window" eight registers are considered to be input registers, eight local registers and eight output

registers. Whenever a subroutine or procedure is run the register window will shift sixteen registers. The previous input and local registers will be hidden, the earlier output registers will become the new input register and sixteen new local and output registers will be accessible. Each subsequent procedure call will result in the register window being moved forward one step and every finished procedure will move the window back one step. When the last register window is reached, a further call will force the processor to move register data to a much slower memory, to make room [20].

Processor configuration register

The application specific register 17 (%asr17) provides information about how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. The register can be accessed through the RDASR instruction, and has the layout shown below [15]:



Field Definitions:

[31:28]: Processor index. In multi-processor systems, each LEON core gets a unique index to support enumeration. The value in this field is identical to the *hindex* generic parameter in the VHDL model.

value in this field is identical to the *hindex* generic parameter in the VHDL model.

[17]: Clock switching enabled (CS). If set, switching between AHB and CPU frequency is available.

[16:15]: CPU clock frequency (CF). CPU core runs at (CF+1) times AHB frequency.

[14]: Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.

[13]: Single-vector trapping (SVT) enable. If set, will enable single-vector trapping. Fixed to zero if SVT is not implemented. Set to zero after reset.

[12]: Load delay. If set, the pipeline uses a 2-cycle load delay. Otherwise, a 1-cycle load delay i s used. Generated from the *lddel* generic parameter in the VHDL model.

[11:10]: FPU option. "00" = no FPU; "01" = GRFPU; "10" = Meiko FPU, "11" = GRFPU-Lite

[9]: If set, the optional multiply-accumulate (MAC) instruction is available

[8]: If set, the SPARC V8 multiply and divide instructions are available.

[7:5]: Number of implemented watchpoints (0 - 4)

[4:0]: Number of implemented registers windows corresponds to NWIN+1.

Figure 6: Leon3 configuration register

2.3.2.RAM usage

The Leon3 core maps all usage of RAM on either the *syncram_dp* component (dual port) or the *syncram_2p*component (double port). They are both from the technology mapping library TECHMAP, in GRLIB library folder. The component that needs to be used can be configured with generics. The default, and recommended, configuration will use *syncram_dp* [15].

Register file

The register file is implemented by using two synram_2p blocks for all technologies where the regfile_3p_infer constant in TECHMAP.GENCOMP is set to 0. The table below shows the organization of the syncram_2p:

Register windows	Syncram_2p organization
2 – 3	64x32
4 – 7	128x32

8 – 15	256x32
16-31	512x31
32	1024x32

Table 2: syncram_2p sizes for Leon3 register file

If regfile_3p_infer is set to 1, the synthesis tool will automatically infer the register. On FPGA technologies, it can be in either flip-flops or RAM cells, depending on the tool and technology. On ASIC technologies, it will be flip-flops. The amount of flip-flops inferred is equal to the number of registers [15].

Number of flip-flops = ((NWINDOWS *16) + 8) * 32

Instruction Trace buffer

The buffer memory is implemented by the instruction trace buffer that uses four identical RAM blocks named the syncram. The syncram will always be 32-bit wide. The depth will depend on the TBUF generic, which indicates the total size of trace buffer in Kbytes. If TBUF = 1 (1 Kbyte), then four RAM blocks of 64x32 will be used. If TBUF = 2, then the RAM blocks will be 128x32 and so on [15].

Scratch pad RAM

From the configuration menu, if the instruction scratch pad RAM is enabled, a syncram block will be instantiated with a 32-bit data width. The depth of the RAM will correspond to the configured scratch pad size. An 8 Kbyte scratch pad will use a syncram with 2048x32 organization. The RAM block for the data scratch pad will be configured in the same way as the instruction scratch pad [15].

The details about the Leon3 processor configuration options, signal descriptions, and component declaration in the design can be seen in [15].

2.4. The GRLIB IP library

The GRLIB IP library is an integrated set of reusable IP cores, designed for system-on-chip (SoC) development. It is developed by AeroFlex Gaisler. The IP cores in the library are centered around the common on-chip bus, and use a coherent method for simulation and synthesis. The library is provided under the GNU GPL license. The library is vendor independent, with support for different CAD tools and target technologies. The unique feature of GRLIB IP library is the plug&play method to configure and connect the IP cores without the need to modify any global resources [14].

2.4.1. Available IP cores

The library includes cores for different venders including the AMBA AHB/APB control, the Leon3 SPARC processor, 32-bit PC133 SDRAM controller, 32-bit PCI bridge with DMA, 10/100/1000 Mbit Ethernet MAC, 8/16/32-bit PROM and SRAM controller, 16/32/64-bit DDR/DDR2 controllers, USB-2.0 host and device controllers, CAN controller, TAP controller, SPI, I2C, ATA, UART with FIFO, modular timer unit, interrupt controller, and a

32-bit GPIO port. The memory and pad generators are also available for Virage, Xilinx, UMC, Atmel, Altera, Actel and Lattice [14].

2.4.2.Library Organization

The library organization in the GRLIB IP library is very systematic in such a way that typically each VHDL library contains a number of packages, declaring the exported IP cores and their interface types. The simulation and synthesis scripts are created automatically by a global 'makefile'. Adding and removing of libraries and packages can be made without modifying any global files, to ensure that the modification of one vendor's library will not affect other vendors. A few global libraries are also provided to define shared data structures and utility functions [14].

2.4.3.Design Concept

In the GRLIB IP library all GRLIB cores use the same data structures to declare the AMBA interfaces, and can then easily be connected together. An AHB bus controller and an AHB/APB bridge are also available in the GRLIB library that allows to assemble quickly a full AHB/APB system [14].

2.4.4.On-chip Bus interconnection

The GRLIB is designed to be "bus-centric", i.e. it is assumed that most of the IP cores are connected through an on-chip bus. The AMBA-2.0 AHB/APB bus has been selected as the common on-chip bus, due to its market dominance (ARM processors) and because it is well documented and can be used for free without license restrictions [14].

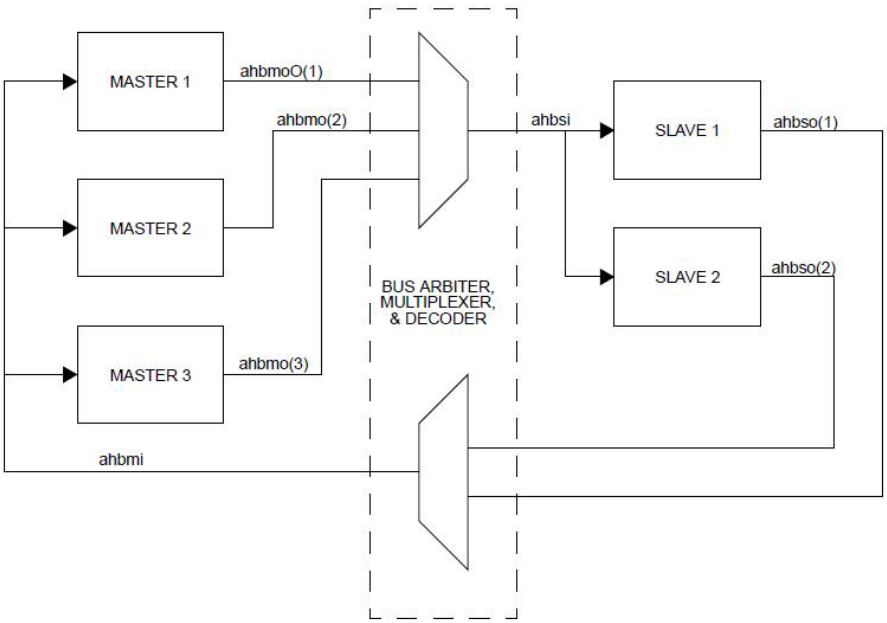


Figure 7: AHB Interconnection view

Since the AHB bus is multiplexed, i.e., it has no tristate signals. Each master drives a set of signals grouped into a VHDL record called HMSTO. The output record of the current bus master is selected by the bus multiplexers and sent to the input record (ahbsi) of all AHB slaves. The output record (ahbso) of the active slave is selected by the bus multiplexer and

forwarded to all masters. A combined bus arbiter, address decoder and bus multiplexer controls which master and slave are currently selected. A view of the bus and the attached units is shown in figure 7 [14].

2.4.5. AHB Slave Interface

The inputs and outputs of AHB slaves are defined as two VHDL records types and are exported through the TYPES package in the GRLIB AMBA library. The elements in the record types correspond to the AHB slave signals as defined in the AMBA 2.0 specification, with the addition of five sideband signals: HBSEL, HCACHE, HIRQ, HCONFIG and HINDEX [14].

```
-- AHB slave inputs
type ahb_slv_in_type is record
  hsel : std_logic_vector(0 to NAHBSLV-1); -- slave select
  haddr : std_logic_vector(31 downto 0); -- address bus (byte)
  hwrite : std_ulogic; -- read/write
  htrans : std_logic_vector(1 downto 0); -- transfer type
  hsize : std_logic_vector(2 downto 0); -- transfer size
  hburst : std_logic_vector(2 downto 0); -- burst type
  hwdata : std_logic_vector(31 downto 0); -- write data bus
  hprot : std_logic_vector(3 downto 0); -- protection control
  hready : std_ulogic; -- transfer done
  hmaster : std_logic_vector(3 downto 0); -- current master
  hmastlock : std_ulogic; -- locked access
  hbsel : std_logic_vector(0 to NAHBCFG-1); -- bank select
  hcache : std_ulogic; -- cacheable
  hirq : std_logic_vector(NAHBIRQ-1 downto 0); -- interrupt result bus
end record;

-- AHB slave outputs
type ahb_slv_out_type is record
  hready : std_ulogic; -- transfer done
  hresp : std_logic_vector(1 downto 0); -- response type
  hrdata : std_logic_vector(31 downto 0); -- read data bus
  hsplit : std_logic_vector(15 downto 0); -- split completion
  hcache : std_ulogic; -- cacheable
  hirq : std_logic_vector(NAHBIRQ-1 downto 0); -- interrupt bus
  hconfig : ahb_config_type; -- memory access reg.
  hindex : integer range 0 to NAHBSLV-1; -- diagnostic use only
end record;
```

A typical AHB slave in GRLIB has the following definition [14].

```
library grlib;
use grlib.amba.all;
library ieee;
use ieee.std_logic.all;

entity ahbslave is
	generic (
		hindex : integer := 0); -- slave bus index
	port (
		reset : in std_ulogic;
		clk : in std_ulogic;
		ahbsi : in ahb_slv_in_type; -- AHB slave inputs
		ahbso : out ahb_slv_out_type -- AHB slave outputs
		);
end entity;
```

The input record is routed to all slaves, and includes the select signals for all slaves in the vector ahbsi.hsel, it is represented by ahbsi. An AHB slave must therefore use a generic that specifies which HSEL element to use. This generic is of type integer, and typically called HINDEX. The output record is represented by 'ahbso' [14].

2.4.6. AHB bus Index Control

The AHB master and slave output records contain the sideband signal HINDEX. This signal is used to verify that the master or slave is driving the correct element of the ahbso/ahbmo buses. The generic HINDEX that is used to select the appropriate HGRANT and HSEL is driven back on ahbmo.hindex and ahbso.hindex. The AHB controller then checks that the value of the received HINDEX is equal to the bus index. The HINDEX and ahbso must always have the same index. An error is issued during simulation if a mismatch is detected [14].

2.4.7. The Plug&Play capability

The system hardware configuration can be detected through the software by using the GRLIB plug&play capability. Such capability makes it possible to use software application or operating systems which automatically configure themselves to match the underlying hardware. Thus the development of the software is greatly simplified, since they do not need to be customized for each particular hardware configuration [14].

In GRLIB, the plug&play information consists of three items:

- A unique IP core ID
- The AHB/APB memory mapping
- The used interrupt vector

This information is sent as a constant vector to the bus arbiter/decoder, where it is mapped on a small read-only area in the top of the address space. Any AHB master can read the system configuration using standard bus cycles, and a plug&play operating system can then be supported [14].

In order to provide the plug&play information from the AMBA units in a harmonized way, a configuration record for AMBA devices has been defined. The configuration record consists of eight 32- bit words, where four contain configuration words defining the core type and interrupt routing, and four contains the 'bank address registers', defining the memory mapping. The configuration word for each device includes 'vendor ID', 'device ID', 'version number', and 'interrupt routing' information. The BARs contain the start address for an area allocated to the device, a mask defining the size of the area, information whether the area is cacheable or pre-fetchable, and a type declaration identifying the area as an AHB memory bank, AHB I/O bank or APB I/O bank. The configuration record can contain up to four BARs and the core can thus be mapped on up to four distinct address areas [14].

2.4.8. Portability

The portability support is provided for components such as single-port RAM, two-port RAM, dual-port RAM, single-port ROM, clock generators and pads. The portability is implemented by means of virtual components with a VHDL generic to select the target technology. In the architecture of the component, VHDL generate statements are used to instantiate the

corresponding macro cell from the selected technology library. For RAM cells, generics are also used to specify the address and data widths, and the number of ports. Same is the procedure for RAM cells when the components are instantiated that have RAM instantiation in them [14].

2.4.9.AHBRAM - Single-port RAM with AHB interface

The AHBRAM core implements a 32-bit wide on-chip RAM with an AHB slave interface. Memory size is configurable in binary steps through a VHDL generic. Minimum size is 1KB and maximum size is dependent on target technology and physical resources. Read accesses are zero-waitstate, write access have one waitstate. The RAM supports byte and half word accesses, as well as all types of AHB burst accesses. Internally the AHBRAM instantiates four 8-bit wide SYNCRAM blocks the details can be seen in [15] and [14].

2.4.10. AHBROM - Single-port ROM with AHB interface

The AHBROM core implements a 32-bit wide on-chip ROM with an AHB slave interface. Read accesses take zero waitstates, or one waitstate if the pipeline option is enabled. The ROM supports byte- and half-word accesses, as well as all types of AHB burst accesses [15].

2.4.11. SYNCRAM_DP - Dual-port RAM generator

The dual-port RAM generator has two independent read/write ports. Each port has a separate address and data bus. All inputs are latched on the rising edge of clk. The read data appears on output directly after the clock rising edge. Address width, data width and target technology is parametrizable through generics. The simultaneous write to the same address is technology dependent, and generally not allowed [15].

2.5. Wrapper

2.5.1. Overview

In the bus-based design approach if the IP components needs to communicate to one or more buses they are usually interconnected by bus bridges. Since the bus specification can be standardized, libraries of components can be developed whose interfaces directly matches these specifications. Companies offer very rich component libraries and specialized development and simulation environments for designing systems around their buses. Even if components follow the bus standard, very simple bus interface adapters may still be needed. Wrappers are used to translate the bus based communication signals, for components that do not directly match these specifications. Although the standard may support a wide range of functionalities, each component may have an interface containing only the functions that are relevant for it. The wrappers are also required if the IP components are compliant to a bus-independent and standardized interface and thus are directly connected to each other. These components may also be interconnected through a bus, in which case standard wrappers can adapt the component interface to the bus [21].

Below is the conceptual diagram showing the wrapper for the AMBA and Avalon bus system. In the figure the Avalon bus based system is wrapped around by the AMBA bus. All the inputs are translated to Avalon and then translated again to the AMBA output.

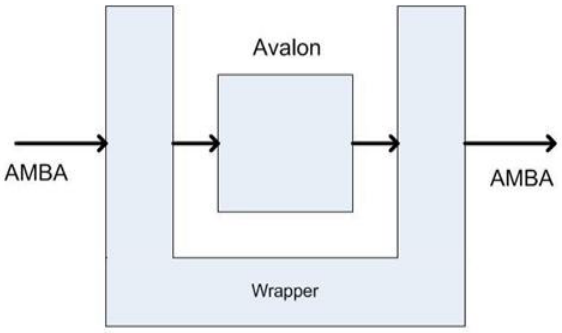


Figure 8: The wrapper concept

2.6. The Hardware

The hardware that is required to develop and test the Leon3 based system is, the Nios development board, Stratix II edition. It provides a hardware platform based on Altera Stratix II device. The connection of the Leon3 processor with the Altera development board is from its template folder in GRLIB. The board number can be easily seen from:

```
"grlib-gpl-1.0.22-b4075 \ boards \ altera-ep2s60-sdr \ Makefile.inc"
```

If it matches with the FPGA number on the board it means that the template folder belongs to that development board.

The Nios development board, Stratix II Edition provides the following features [7]:

- A Stratix II EP2S60F672C5ES device with 24,176 adaptive logic modules (ALM) and 2,544,192 bits of on-chip memory
- 16 Mbytes of flash memory
- 1 Mbyte of static RAM
- 16 Mbytes of SDRAM
- On board logic for configuring the Stratix II device from flash memory
- On-board Ethernet MAC/PHY device
- Two 5V-tolerant expansion/prototype headers each with access to 41 Stratix II user I/O pins
- CompactFlashTM connector for Type I CompactFlash cards
- Mictor connector for hardware and software debug
- Two RS-232 DB9 serial ports
- Four push-button switches connected to Stratix II user I/O pins
- Eight LEDs connected to Stratix II user I/O pins
- Dual 7-segment LED display
- JTAG connectors to Altera® devices via Altera download cables
- 50 MHz oscillator and zero-skew clock distribution circuitry
- Power-on reset circuitry

For the further details about the board see the board manual [7]

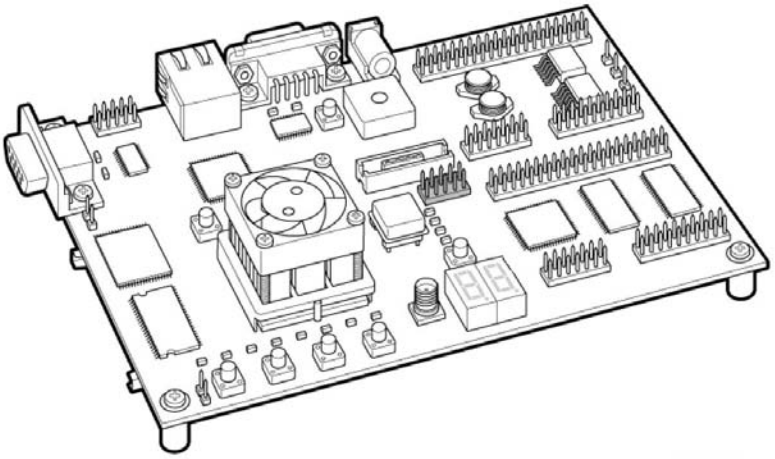


Figure 9: Nios II development board Stratix II edition

2.7. GUI Development Environment

The GUI for the generation of Leon3 based 2x2 NoC system is developed in Microsoft Visual Basic 2008. The Microsoft Visual Basic 2008 features the .NET framework and shell function that is used in the GUI development.

2.7.1. Visual Basic 2008

The Visual Basic 2008 is a development tool that can used to build software applications that perform useful work and look great within a variety of settings. By using Visual basic 2008, various applications for the Windows operating system can be created, e.g., the Web, hand-held devices, and a host of other environment and settings. Comparing to the other available platforms the interfacing of the Visual Basic with other environment is quite easy. Many applications utilize this important advantage of Visual Basic to increase the productivity in the daily development work [28].

2.7.2. The Graphical User Interface

The Microsoft Windows uses a graphical user interface, or GUI (pronounced “gooey”). The Windows GUI defines how the various elements look and function. For a Visual Basic programmer, there is available a toolbox of these elements to create new windows, called *forms*. There is a toolbox to add the various elements, called *controls*. The project will be written follow a programming technique called *object-oriented programming (OOP)* [29].

2.7.3. The .NET Framework

The programming languages in Visual Studio run in the .NET Framework. The Framework provides for easier development of Web-based and Windows-based applications, allows objects from different languages to operate together, and standardizes how the languages refer to data and objects. The .NET languages that includes the visual basic also, compile to (are translated to) a common machine language, called Microsoft Intermediate Language (MSIL). The MSIL code, called *managed code*, runs in the Common Language Runtime (CLR), which is part of the .NET Framework [29].

2.7.4.VB Shell Function

The shell function is used to run the executable programs from within a Visual Basic program. The shell takes two arguments, the first one is the name that includes the path of the executable to run, and second is the argument that specifies the window style of the program. The shell function runs the target programs asynchronously, so that the other program can be started without finishing the first one. The details about the shell function can be found in [30].

Chapter 3

3. Network on Chip

This chapter will present the basic concept about the Network on Chip and some common terminologies that will be used while discussing the NoC system. Later in the chapter there will be introduction of the 2D Mesh 2x2 NoC system, and also the address decoding, packet format and switch architecture will also be discussed.

3.1. Basic NoC

The NoC architecture is based on a network of switches also called nodes. Each switch is connected to another by mean of some physical links like wires. The connections between switches can be of different topologies. The most commonly used topology is the Mesh topology, in which the nodes are arranged in the form of a mesh like in network of computers. In Mesh topology every switch is connected to every other switch using links. The communication is done by routing packets over the network instead of driving dedicated wires. Each switch is further connected to a resource, the resource is equipped with an interface called resource network interface (RNI). The figure shows the 3x3 mesh topology [12].

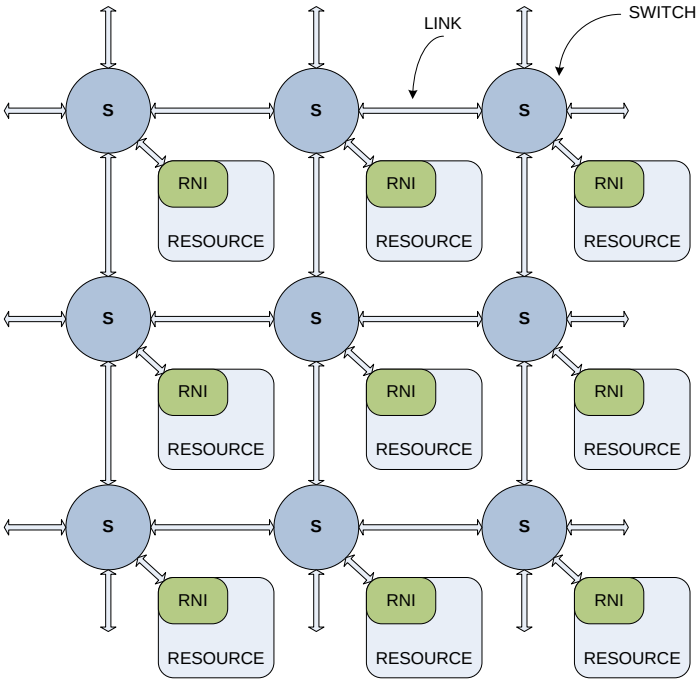


Figure 10: The 9 nodes Mesh NoC

Some common terminologies that are used in NoC design architectures are described below.

3.1.1.Resource

A resource is the unit that is connected to the switch. It can be a processor, memory, IP block, FPGA, ASIC, or a bus-based sub-system [12]. For the current thesis the resource is the Leon3 processor.

3.1.2.Resource Network Interface (RNI)

The resource network interface (RNI) is used to connect the resource to the rest of the network through the switch. The main purpose of the RNI is to translate the communication protocol used by the resource into the network communication protocol. Thus the RNI is responsible for translation, message fragmentation, packet formatting, packet flits reordering and any other specific requirement that may be set by the resource communication interface [12].

3.1.3.Links

The physical communications by wires, between the two switches are termed as a links. Links are bi-directional and their width may vary from one NoC design to the other depending on the flit size and/or the hardware limitations [12].

3.1.4.Switch

The switch is the basic unit in NoC design. The communication network is created by connecting the switches with one another. Each switch represents a node and is connected to a resource. Resources communicate with other resources through the switch network. The concept of switch interfacing with the resource through network interface is shown below [12] [22].

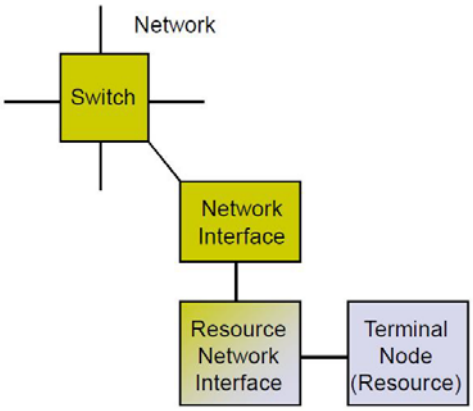


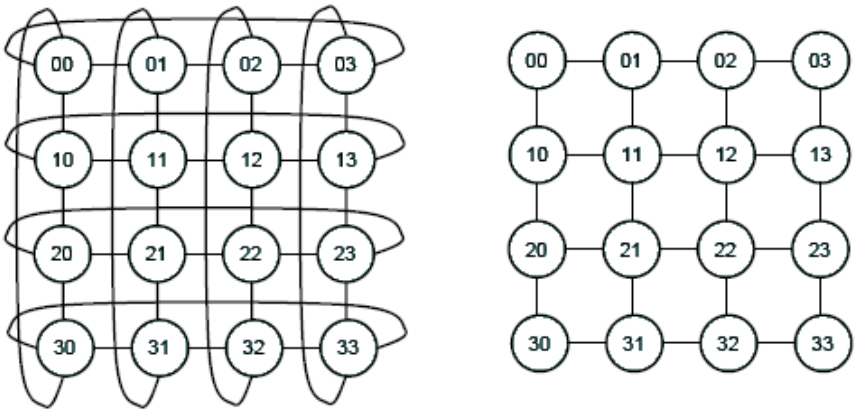
Figure 11: Node representation

3.1.5.Network Topology

The term network topology is used to describe the static arrangement of nodes and channels in the network. There are different ways the nodes and channels can be connected to each other to form a Network on Chip. Depending on constraints and requirements, the network can be divided into two main categories, based on its topology [3].

Direct Network

In direct network topology every node is both a terminal and a switch, like in torus and mesh network. Below is shown the 4x4 Torus and Mesh network diagrams [3].

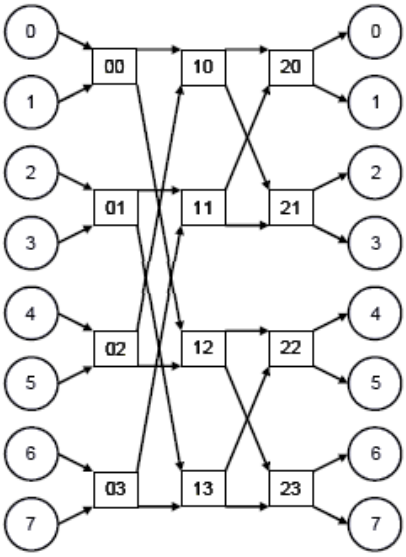


"4x4"-Torus

"4x4"-Mesh

Indirect Network

In the indirect network topology a node is either a terminal or a switch, like in case of butterflies, as shown below [3].



Butterfly

3.1.6.Flow Control

The flow control determines how the network resources, such as channel bandwidth, buffer capacity etc., are allocated to the packets present in the network. If the two packets in the network want the same channel, the flow control will allow only one of the packets to use that channel at that time, and then also deal with the other one. So the flow control determines how the resources are to be used for the traversing of the packets. The resources might be the 'channel bandwidth' or the 'buffer capacity' [12].

Based on the resource requirement, the flow control can be divided into two main types:

Buffer-less flow control

In buffer-less flow control the packets are either dropped or misrouted because there are no buffers to store them [3].

Buffered flow control

In buffered flow control the packets that cannot be routed via the desired channels are stored in buffers [3].

3.1.7. Routing Algorithm

The routing algorithm is responsible for finding a path from the source node to the destination node within a given topology. The term routing means selecting a path from a source node to a destination node in a particular topology. Routing algorithms can be classified as follows [23].

Deterministic algorithms

They always choose the same path between two nodes, even if there are multiple possible paths. The deterministic routing algorithms are easy to implement and can be made deadlock free. The problem with these algorithms is that they do not use path diversity and thus bad on load balancing [23].

Oblivious algorithms

The oblivious algorithms always choose a route without knowing about the state of the network. So it can be said that all random algorithms are oblivious algorithms. Therefore all deterministic algorithms are oblivious algorithms [23].

Adaptive algorithms

The adaptive algorithms use information about the state of the network to make routing decisions. The information includes the length of queues and the historical channel load [23].

Minimal and non-minimal algorithms

Two terms are commonly used when the routing algorithms are discussed; these are the minimal and the non-minimal algorithms. As evident from their names the minimal algorithms always choose the shortest path between two nodes where as the non-minimal algorithms allow the non-minimal paths also [23].

3.2. 2D Mesh 2x2 NoC

3.2.1. Overview

The implemented NoC design is a 2DMesh 2x2 network, each node is mapped to a single resource (Leon3 processor). The designed NoC uses Mesh topology as shown in figure below. The platform uses ‘Buffer-less Flow Control’, that is the switches are not equipped with any buffers. Thus, incoming flits are not stored and must be routed out at the same rate at which they arrive. The NoC platform utilizes ‘Dimension-Order Routing’. Every node in the

network is assigned a unique address. The digits of the destination node addresses are used to route the packet through the network. The designed NoC platform uses a y-before-x routing algorithm. In the 'xy' routing algorithm when a flit reaches a switch, the switch compares the destination node address with its local node address. Based on the result of the comparison, if the flit is required to be routed in both X and Y dimensions, it is first routed along the Y-dimension path. Once the flit is in the desired Y-dimension (desired row), the routing required along the X-dimension path is carried out [12].

Below is the structural view of the 2x2 NoC design. All the switches connected to the resources through RNI and with one another using links [12].

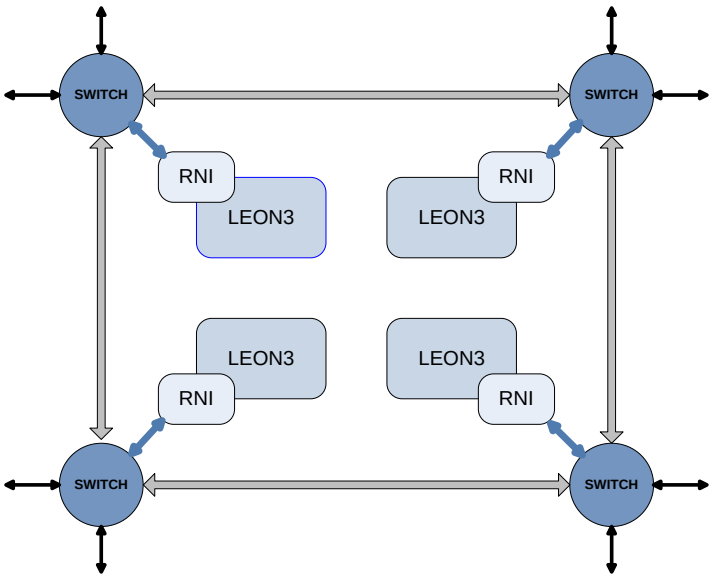


Figure 12: 2x2 NoC pattern

3.2.2.Switch architecture

Each node or switch has total of five ports of shown in figure 12, out of these five ports two are used to connect to the nearby switches and one port is for interfacing with the resource. The two remaining nodes are kept in the design for future use, but marked as busy so that the packets are not routed to those directions. The details about the switch architecture can be seen in [4] and [12].

3.2.3.Node Address Decoding

In the implemented NoC design, the node IDs have been assigned using a 4-bit frame, 2 bits for the row position and 2 bits for the column position. In this way the node in the lower left corner has row 0 and column 0, thus has been given the ID '00, 00' (row, col), and in the same way with the other nodes also, as shown in figure 13 [12].

The 4 bit frame will thus enable the design implementation of 4 nodes in a 2x2 fashion. With the frame size of 4 bit frame, the design can be extended to the 4x4 NoC. These node IDs have been used as a routing decision base. In every node, the node ID is hard coded enabling it to recognize packets sent to it and also to make routing decisions accordingly [12].

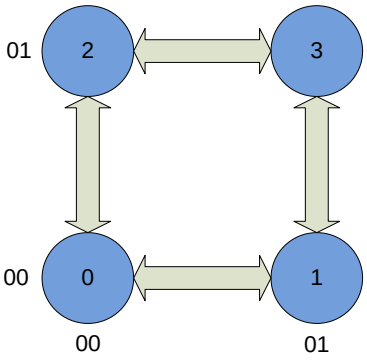


Figure 13: 2x2 NoC node IDs

3.2.4. The Possible Routing directions

The ports in the NoC design are bidirectional, i.e., the packets can move simultaneously in both directions. The figure below shows all possible routing directions. As shown, the nodes used for the transferring of the data are categories with respect to their orientation in xy plane. In doing so the Node 10 is called as Upper Left (UL), Node 11 as Upper Right (UR), Node 00 is Lower Left (LL) and the Node 01 as Lower Right (LR) in the design coding.

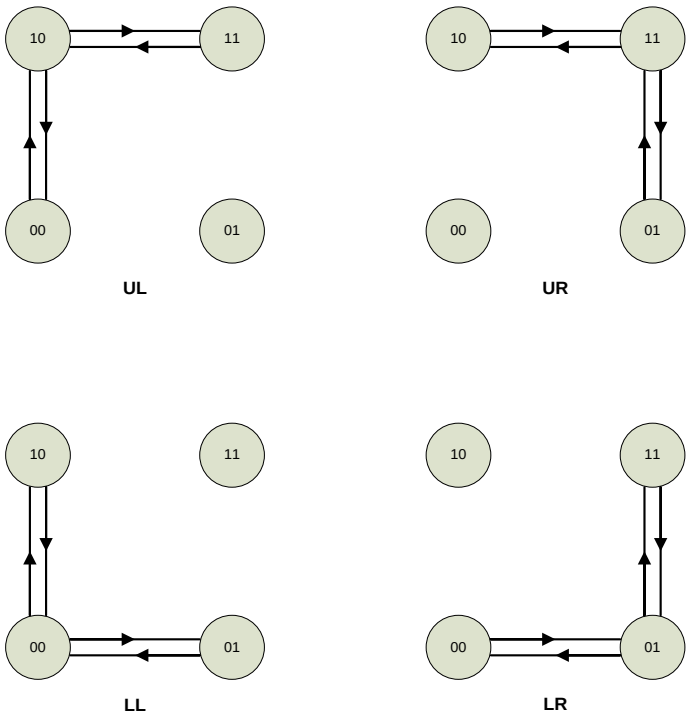


Figure 14: Possible Packet routing

3.2.5. The Packet Format

The flits have 52 bits in total, in those 52 bits the 32 bits are for data and rest 20 bits are for the header. The header contains the additional information for the packet transfer from source to destination like the setup information for receiver for successful reception and time stamp for debugging etc. as shown below [12].

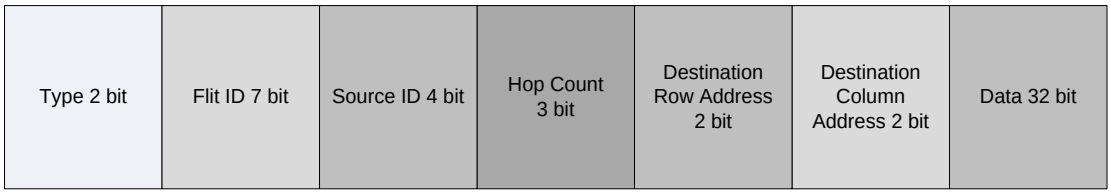

Type 2 bit
Flit ID 7 bit
Source ID 4 bit
Hop Count 3 bit
Destination Row Address 2 bit
Destination Column Address 2 bit
Data 32 bit

Figure 15: 2x2 NoC Packet format

The use of each bit is given in the package file of the NoC design project, named the “NoC parallel package”. The part of the code given below describes the packet format with respect to number of bits. The header size is 20 bits and data size is 32 bits as shown below.

```
-- Size, type and bit locations of all fields in the packet format are
defined here

constant Type_size:integer:=2; -- Empty=0-, Valid=1-, Setup=11, Data=10
constant Flit_id_size:integer:=7; -- for 0-127
constant Src_size:integer:=4; -- New for 2x2: Source address/channel number
constant HC_size:integer:=3; -- Hop Counter
constant UD_size:integer:=2; -- Up/Down Max 2x2 NoC
constant LR_size:integer:=2; -- Left/Right Max 2x2 NoC
constant
Header_size:integer:=Type_size+Flit_id_size+Src_size+HC_Size+UD_Size+LR_size
;
constant Data_size:integer:=32; -- Data field
```

the details about the packet format can be found in [4].

Chapter 4

4. Design Setup

This chapter will provide the basic understanding about the GRLIB IP library cores and familiarization with the GRTools. It is also provides the information about the working of the Leon3 processor and the plug&play configurations. The procedures discussed here will be the baseline for the development of Leon3 based 2x2 NoC design.

4.1. Requirements

The following hardware and software components are required in order to use and implement the Leon3 system that is based on a template design, leon3-altera-ep2s60-sdr:

- GRLIB IP Library grlib-gpl-1.0.22-b4075
- PC work station with Linux or Windows XP/Vista with GRTools and Cygwin
- Altera's Startix II board with USB JTAG programming cable
- Mentor Graphics Modelsim SE 6.5c
- Altera's Quartus II 7.2

4.2. GRLIB Installation

GRLIB is distributed as a zipped file and can be installed in any location on the host system, once unzipped; the distribution in the folders can be seen with the following file hierarchy.

Folder Name	File hierarchy description
Bin	various scripts and tool support files
Boards	support files for FPGA prototyping boards
Designs	template designs
Doc	Documentation
Lib	VHDL libraries
Netlists	vendor specific mapped netlists
Software	software utilities and test benches
Verification	test benches

Table 3: GRLIB folders

The GRLIB uses the GNU 'make' utility to generate scripts and to compile and synthesize designs. The library should be installed on a UNIX system or in a 'Unix-like' environment. Therefore on PC platform they are suitable for Linux and Windows with Cygwin [14].

4.2.1. Directory Organization

The GRLIB IP library is organized around the VHDL libraries, where each IP vendor is assigned a unique library name. Each vendor is also assigned a unique subdirectory under grlib/lib in which all vendor-specific source files and scripts are contained. The vendor-specific directory can contain subdirectories, to allow for further partitioning between IP cores etc.

The basic directories delivered with GRLIB under grlib-1.0.x/lib are as given below [14]:

Library Name	Description
grlib	packages with common data types and functions
gaisler	Gaisler Research's components and utilities
tech/*	target technology libraries for gate level simulation
techmap	wrappers for technology mapping of marco cells (RAM, pads)
work	components and packages in the VHDL work library

Table 4: GRLIB directory organization

4.2.2. Host platform support

GRLIB is designed to work with a large variety of hosts. As a baseline, the following host software must be installed for the GRLIB configuration scripts to work [14].

- Bash shell
- GNU make
- GCC
- Tcl/tk-8.4/8.5

4.3. GRTools

The GRTools is a single file that is an installer for the Windows. It is installed with the following tools in a uniform way [32].

- BCC cross-compiler
- RCC cross-compiler including RTEMS-4.10
- LEON IDE including Eclipse and CDT
- GRMON
- GrmonRCP
- TSIM-LEON3
- HASP HL drivers for GRMON and TSIM
- Development tools:
    - MSYS
    - MinGW
    - MSYS DTK
    - Autoconf
    - Automake

The details can be found in [27], [26] and [13].

4.3.1. Windows with Cygwin

Although the GRTools provides the complete windows installation kit, however there are few versions that are not fully compliant with the Gaisler's GRTools, so there is need for the installation of the Cygwin along with MinGW. Note that gcc and the make utility must be selected during the Cygwin installation. Some versions of Cygwin are known to fail due to a broken 'make' utility. , The make utility and associated scripts will work, although somewhat slow [14].

4.4. The Working of GRTools

Before beginning it is necessary to make sure that the GRTools are installed properly. To determine if the installation done properly or not, use the “make” command at the command prompt from the template folder. The following line appears on the console means that the installation done properly.

```
sparc-elf-gcc –I../../software/leon3 –ffast-math -03 –c
../../software/leon3/fpu.c
```

The details about the leon3 simulation and bcc can also be found in [25] and [26] respectively

4.5. Implementation

Implementing a simple Leon3 system, to check the working of GRTools and for basic understanding, is done by using one of the template designs in the ‘designs’ directory. For this thesis the leon3-altera-ep2s60-sdr design is used. Implementation is typically done in three basic steps [14]:

i. Configuration of the design using xconfig
ii. Simulation of design and test bench
iii. Synthesis and place&route

The template design is based on the following files already available, below is the brief description of each.

config.vhd
It is a VHDL package that contains the design configuration parameters. The file is automatically generated by the make xconfig. Each core in the template design is configurable using VHDL generics. The value of these generics is assigned from the constants declared in config.vhd, created with the xconfig GUI tool [14]

leon3mp.vhd
It contains the top level entity and instantiates all on-chip IP cores. It uses config.vhd to configure the instantiated IP cores. It is the main design file for Leon3 core [14].

ahbrom.vhd
It is a VHDL file that contains the parameters for the booting the processor, from a fixed address location [14].

testbench.vhd
It is the testbench with external memory, emulating the Altera’s EP2S60-SDR board [14].

4.6. AHB plug&play configuration

The GRLIB implementation of the AHB bus includes a mechanism to provide plug&play support. The plug&play support consists of three parts:

- The identification of attached units (masters and slaves)
- Address mapping of slaves
- The interrupt routing.

The plug&play information for each AHB unit consists of a configuration record containing eight 32-bit words. The first word is called the identification register and contains information on the device type and interrupt routing. The last four words are called bank address registers, and contain address mapping information for AHB slaves. The remaining three words are currently not assigned and could be used to provide the core-specific configuration information [14].

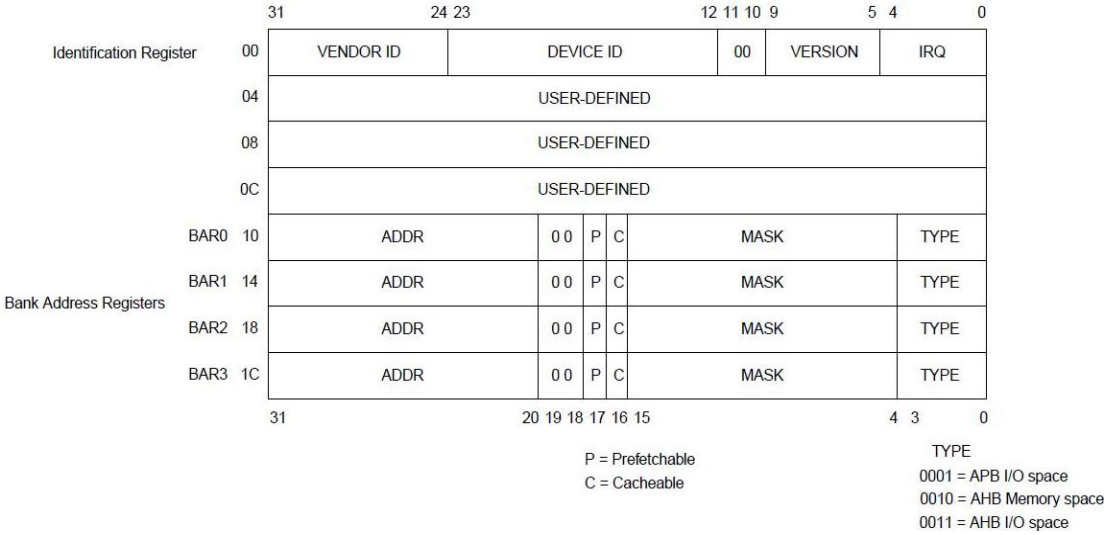


Figure 16: The plug&play configuration

The plug&play information for all attached AHB units appear as a read-only table mapped on a fixed address of the AHB (typically at 0xFFFFF000). The configuration records of the AHB masters appear in 0xFFFFF000 - 0xFFFFF800, while the configuration records for the slaves appear in 0xFFFFF800 - 0xFFFFFFFC. Since each record is 8 words (32 bytes), the table has space for 64 masters and 64 slaves. A plug&play operating system (or any other application) can scan the configuration table and automatically detect which units are present on the AHB bus, how they are configured, and where they are located (slaves) [14].

The configuration record from each AHB unit is sent to the AHB bus controller via the HCONFIG signal. The bus controller creates the configuration table automatically, and creates a read-only memory area at the desired address (default 0xFFFFF000). Since the configuration information is fixed, it can be efficiently implemented as a small ROM or with relatively few gates [14].

4.6.1.Device identification

The Identification Register contains three fields to identify uniquely an attached AHB unit: the vendor ID, the device ID, and the version number. The vendor ID is a unique number assigned to an IP vendor or organization. The device ID is a unique number assigned by a vendor to a specific IP core. The device ID is not related to the core's functionality. The version number can be used to identify (functionally) different versions of the unit. The vendor IDs are declared in a package in each vendor library, usually called DEVICES. Vendor IDs are provided by Gaisler Research. Vendor ID 0x00 is reserved to indicate that no core is present. The unused slots in the configuration table will have Identification Register set to 0 [14].

4.7. Leon3 Configuration

The leon3 configuration procedure is described as below. In GRLIB, the graphical configuration is started with the “make xconfig” command. The main menu of the configuration is shown in figure below. The settings made for a GRLIB system are stored in the file “config.vhd”. It can be edited manually or through the graphical interface. The values from the configuration file are then referenced by the top design file, “leon3mp.vhd”, where all components are instantiated [11].

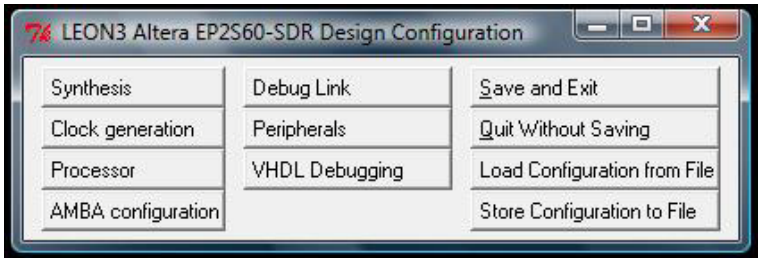


Figure 17: Leon3 Design Configuration

4.7.1. Synthesis

For Altera FPGAs there is a single option to cover all of the Altera families. There is an “inferred” option which specifies generic memories and pads to let a synthesis with the right capabilities automatically. It is also possible to let the synthesis tool handle the insertion of RAM and pads on its own by setting “Infer RAM” and or “Infer pads” to “yes”. There is an option to “Disable asynchronous reset” for which the whole design can be completely reset at any time during a clock cycle if the target technology does not support it. To enable the designed RNI memory only the infer RAM is set to ‘yes’, rest all are set to default ‘no’ [20].

Figure 18: Synthesis Menu

4.7.2. Clock Generation

Some target technologies have dedicated circuitry such as Phase-Locked Loop (PLL) to manage the system clock and the SDRAM clock. This may allow phase adjustment and resynchronization of the clock signal. The supported FPGA clock generators for Altera Stratix, Stratix2, Cyclone and Cyclone2 support the Altera ALTPLL for clock management. The settings are left as default for current thesis as shown below [20].

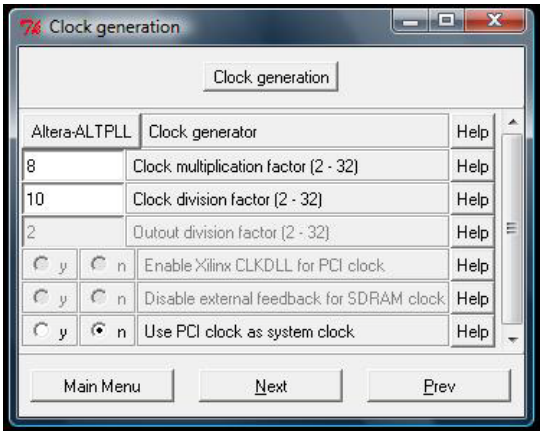


Figure 19: Clock generation

4.7.3.Processor

The Leon3mp design can accommodate up to 4 Leon3 processor cores. The default is 1, that allows only one Leon3 processor on an AMBA bus. If more than one Leon3 is required then the system can be called as SMP because identical Leon3 cores connected to the same AMBA bus. The processors are fully independent and can communicate through shared memory. The caches are of “write through” type which means that data will always be written directly to memory even if that part of the memory is already present in the cache [11] [20].

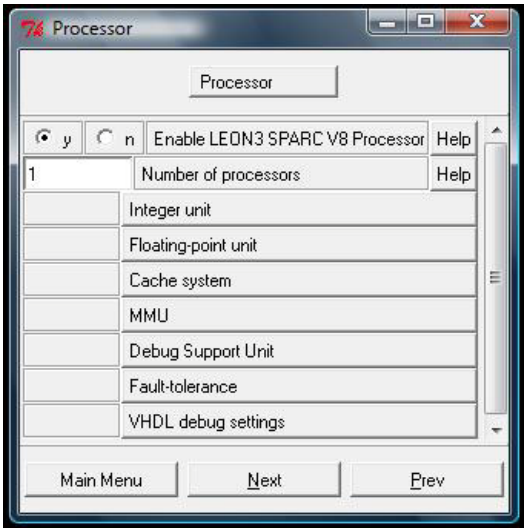


Figure 20: Processor Menu

4.7.4.Integer Unit

The Leon3 integer unit implements full SPARC V8 standard, that includes the hardware multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC standard (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture) [11].

In the SPARC V8 instruction set the multiplication and division instructions are optional. Implementing those instructions in hardware will greatly speed up such calculations at the cost of the equivalent of about 8000 logic gates. If the hardware multiply unit becomes a bottleneck (critical path) for the overall clock speed of the system its workload can be divided

in 5 smaller steps instead of the default of 4. This will increase the latency of the multiply and divide instructions to 5 clock cycles. Another possible critical path is the Load instruction. By default there is a one clock cycle delay between a Load instruction being run and the data being available. If the target technology results in slow data cache memory, increasing this delay to two cycles may allow for higher operating frequencies. To reduce power consumption, a power-down mode can be included. It can save power by shutting down the integer pipeline and the caches, when they are not being used. The integer pipeline will stay in sleep mode until there is an interrupt [20].

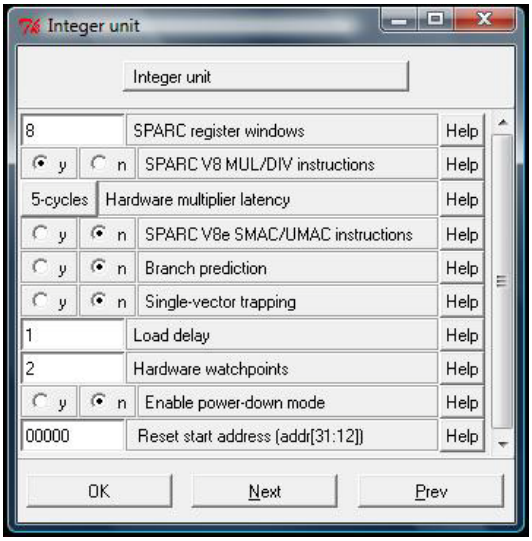


Figure 21: Integer unit menu

4.7.5. AMBA Configuration

The AMBA AHB bus works on the master-slave principle. There are some devices on the bus called ‘master’, which are allowed to initiate data transfers. While some are ‘slave’, that can only respond to data requests made by a master. When there are several masters competing to use the bus at the same time there is an ‘arbiter’ unit to decide which one can go first. The AMBA implementation in GRLIB has two algorithms do this. The default arbitration method is the one which always give priority to the master with the highest bus index. The second method is “round-robin” where all masters are given equal chance to use the bus in turn [20].

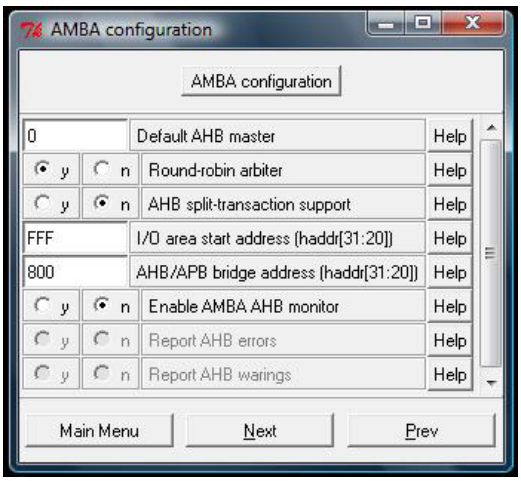


Figure 22: AMBA Configuration menu

There are two more fields that are left unchanged in the configuration menu, first one is the I/O start address, that selects the MSB address (HADDR[31:20]) of the AHB IO area, as defined in the plug&play extensions of the AMBA bus. The second is the APB/AHB bridge address that selects the MSB address (HADDR[31:20]) of the APB bridge. It should be kept at 800 for software compatibility. The AMBA AHB monitor will check for illegal AHB transactions during simulation and has no impact on the synthesis.

4.7.6.Peripherals

Through the AMBA bus a Leon3mp System-on-Chip can communicate with other Intellectual Property (IP) cores. The GRLIB package contains a number of optional cores that can provide useful functions and interfaces for the On-chip and Off-chip resources such as memories and networks [20].

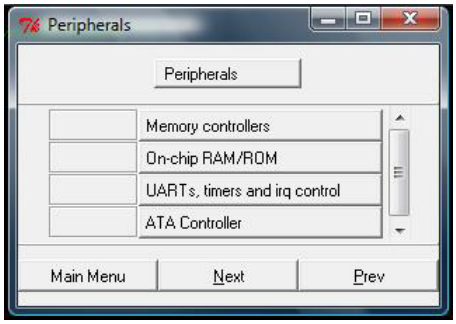


Figure 23: Peripherals menu

4.7.7.On-chip RAM/ROM

From the Peripheral menu the On-Chip ROM/RAM are shown below. The content of the ROM are required as a VHDL-file (ahbrom.vhd) to allow the synthesis directly into the system. The ROM has start address of 000, and the pipelined ROM access is disabled. Similar to the on-chip ROM an on-chip RAM is implemented on the AMBA AHB bus. The size of the on-chip RAM is 4KB and has a starting address of 0x400 for the current design. The on-chip RAM and ROM are directly mapped on the AHB bus and do not require the memory controller [15].

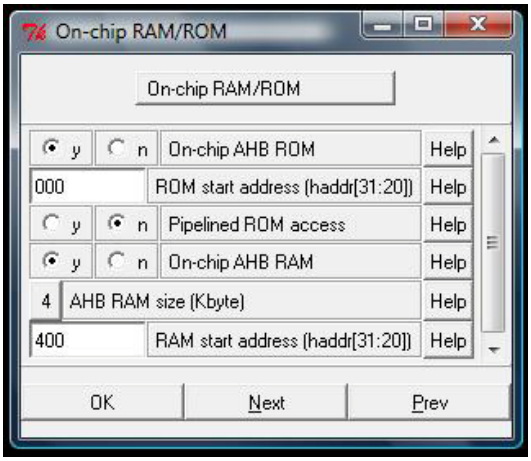


Figure 24: On-chip RAM/ROM peripheral menu

4.8. Simulation

The template design can be simulated in a testbench that emulates the prototype board. For the template design the testbench includes external PROM and SDRAM which are pre-loaded with a test program. The test program will execute on the Leon3 processor, and tests various functionality in the design. The test program will print diagnostics on the simulator console during the execution. The following command should be give to compile and simulate the template design and testbench [14].

```
make vsim
vsim testbench
```

4.9. Synthesis and place&route

The template design can be synthesized with Altera's Quartus II. The synthesis can be done in batch mode as well as interactively.

To use Quartus in batch mode, use the command:

```
make quartus
```

To use Quartus interactively, use the command:

```
make quartus-launch
```

In both cases, the final programming file is called 'leon3mp.bit'. The name of the top design can be changed by modifying the 'Makefile' in the template design [14].

4.9.1. Running applications on target

To load and run the test programs on the Leon3 processor, the '`make soft`' is used that will generate the binaries of the programs in the current template design folder. To download and debug applications on the target board, GRMON debug monitor is used. GRMON can be connected to the target using RS232, JTAG, Ethernet or USB. In this case GRMON connected to the target through USB byteblaster cable [14] [6].

```
grmon-eval -altjtag –u
```

Here the word '`-eval`' is for evaluation version of GRMON. The above command will start a debug monitor program on the console as shown below.

```
GRMON LEON debug monitor v1.1.42 evaluation version

 Copyright (C) 2004, 2005 Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com

 This evaluation version will expire on 3/5/2011
 using Altera JTAG cable
 Selected cable 1 - USB-Blaster [USB-0]
JTAG chain:
@1: EP2S60ES (0x020930DD)

 GRLIB build version: 4075

 initializing
 detected frequency:  40 MHz
```

```
Component                            Vendor
LEON3 SPARC V8 Processor             Gaisler Research
AHB Debug UART                       Gaisler Research
AHB Debug JTAG TAP                   Gaisler Research
LEON2 Memory Controller              European Space Agency
AHB/APB Bridge                       Gaisler Research
LEON3 Debug Support Unit             Gaisler Research
ATA Controller                       Gaisler Research
Generic APB UART                     Gaisler Research
Multi-processor Interrupt Ctrl       Gaisler Research
Modular Timer Unit                   Gaisler Research
General purpose I/O port             Gaisler Research

Use command 'info sys' to print a detailed report of attached cores
```

The debug program initializes the board and shows the attached peripheral units in the current design.

The complete details about the attached peripherals and address locations can found by using the ‘`info sys`’ command, following is the system information of ‘altera-ep2s60-sdr’ default template design.

```
Grmon> Info sys

00.01:003   Gaisler Research  LEON3 SPARC V8 Processor (ver 0x0)
             ahb master 0
01.01:007   Gaisler Research  AHB Debug UART (ver 0x0)
             ahb master 1
             apb: 80000700 - 80000800
             baud rate 115200, ahb frequency 40.00
02.01:01c   Gaisler Research  AHB Debug JTAG TAP (ver 0x0)
             ahb master 2
00.04:00f   European Space Agency  LEON2 Memory Controller (ver 0x0)
             ahb: 00000000 - 20000000
             ahb: 20000000 - 40000000
             ahb: 40000000 - 80000000
             apb: 80000000 - 80000100
             8-bit prom @ 0x00000000
             32-bit static ram: 1 * 1024 kbyte @ 0x40000000
             32-bit sdram: 1 * 16 Mbyte @ 0x60000000, col 8, cas 2,
ref 7.8 us
01.01:006   Gaisler Research  AHB/APB Bridge (ver 0x0)
             ahb: 80000000 - 80100000
02.01:004   Gaisler Research  LEON3 Debug Support Unit (ver 0x1)
             ahb: 90000000 - a0000000
             AHB trace 128 lines, 32-bit bus, stack pointer 0x400ffff0
             CPU#0 win 8, hwbp 2, itrace 128, V8 mul/div, srmmu, lddel
1
                   icache 2 * 8 kbyte, 32 byte/line lru
                   dcache 2 * 4 kbyte, 16 byte/line lru
05.01:024   Gaisler Research  ATA Controller (ver 0x0)
             irq 10
             ahb: fffa0000 - fffa0100
             Device 0: (None)
             Device 0: (None)
             Device 1: (None)
             Device 1: (None)
01.01:00c   Gaisler Research  Generic APB UART (ver 0x1)
             irq 2
             apb: 80000100 - 80000200
             baud rate 38461
02.01:00d   Gaisler Research  Multi-processor Interrupt Ctrl (ver 0x3)
             apb: 80000200 - 80000300
03.01:011   Gaisler Research  Modular Timer Unit (ver 0x0)
             irq 8
```

```
              apb: 80000300 - 80000400
              8-bit scaler, 2 * 32-bit timers, divisor 40
05.01:01a   Gaisler Research  General purpose I/O port (ver 0x0)
              apb: 80000500 - 80000600
```

The ‘sparc-elf-gcc’ command can be used to execute a c-program.

```
sparc-elf-gcc xxxx.c –o xxxx.exe
```

The above command will generate the ‘.exe’ and ‘.o’ files of the program that can be loaded to the GRMON debugger by using ‘load xxxx’ command, when it is loaded the application can be run by using ‘run’ command from the GRMON prompt.

The ‘.exe’ files can be disassembled by using command

```
sparc-elf-objdump -D xxxx.exe > xxxx.disas
```

The ‘xxxx.exe’ executable is converted into motorola srec-format (sram.srec/sdram.srec). These files are then typically used to initialize memory simulation models instantiated in testbench.vhd. The more details can be seen in [13], [25] and [8].

Chapter 5

5. Methodology

This chapter will discuss the methodology followed for the development of Leon3 based 2x2 NoC design. It starts with discussion about the working of RNI and the development of wrapper for translation of signals from Avalon to AMBA, and then there will be description of the procedures to adapt the RNI memory in the system as AHB slave, the modification of AHBROM and the development of the Top Module to integrate Four Leon3 systems. At the last the development of the GUI to configure the new system will be discussed.

5.1. Resource Network Interface

The Resource Network Interface (RNI), interfaces with the resource (the Leon3 processor) through AMBA AHB bus. The RNI is connected to the system as an AMBA AHB slave that allows the RNI to behave like a slave memory. The Leon3 acts as a master in the system, so the RNI waits for the command from the master and performs the required functions. For reading or writing the control information and the data packets, the memory is divided in the form of segments. Each segment is identified by the memory offset that separates them [12].

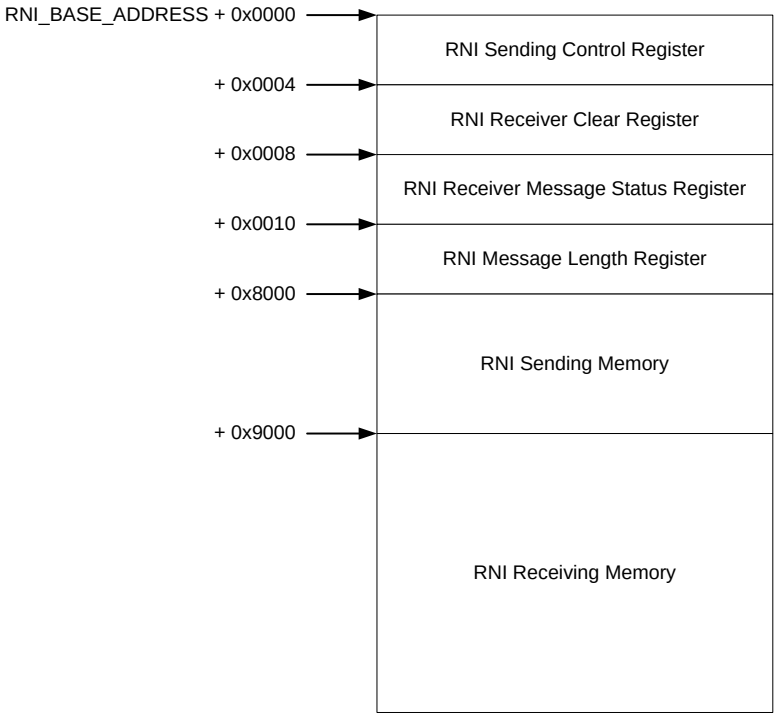


Figure 25: RNI memory configuration

In RNI there are separate read and write memories. The read memory is represented by the read_buffer (1KB) and the write memory by write_buffer (2KB). The write_buffer is double in size than that of the read_buffer. That is for simultaneously receiving the packets from more than one resource. The receive channel allocation is done automatically as the flits are

received by the RNI. A ‘busy_flag’ and a ‘Ch_sourceNo’ is associated with every receive channel. The details about that can be found in [12].

5.1.1. Communication between RNI and Switch

The RNI has two ports named inport and outport that are used to connect the RNI to the switch, they are 50 bits data ports. In addition, there are two control signals, the ‘read_RNI’ and ‘write_RNI’, both control signals are activated and or deactivated by the switch. The switch receives and transmits data flits for every fourth cycle. When the switch route the data to the RNI it insert the write_RNI, whereas read_RNI is inserted when the data flit is to be routed from the RNI to the switch. For the data communication the RNI places the data flit on the data port and waits for the read_RNI, when the switch inserts read_RNI only then it can route the RNI’s inserted flit into the network. By using this mechanism the network saturation will never occur [12].

5.1.2. Routing Management

For the routing management the RNI need to know destination node address, that is specified to it by the resource. The resource (Leon3 processor) acts as a master, whereas the RNI is the slave. Once the command received from the resource, then in its response the packet is generated inside the RNI. The information about the destination ID is placed in the flit header field, the switch uses this information to make the routing decision. In the switch, every time the flit moves from one node to the other, and then it is utilized when making a routing decision, at that point the ‘Hop Count’ field is incremented by one, denoted by ‘HC’. In 2x2 NoC design all the switches are the ‘corner switches’, i.e. the only two of the four ports are utilized and the remaining two ports are provided with a permanent busy signals. The switches are not connected to any off-board switches contrary to the 4x4 NoC design so the routing management is relatively simple [12].

5.2. The wrapper

In the thesis the existing NoC design is wrapped around by the AMBA AHB interface signals. The NoC design that was previously used in the Quadcore NoC design [12] is based on Avalon switch fabric. In order to make that NoC design communicable with the AMBA AHB bus interface a wrapper has been developed. The reason for this is to keep the integrity of the existing designed NoC intact, and utilizing it in the current 2x2 NoC design.

In order to translate the AMBA bus signals to the Avalon and vice versa. The wrapper is implemented in the RNI design file. In RNI design file there is a component called the “nios_2_noc” that has Avalon slave ports; those ports are mapped in the RNI design file to their corresponding AHB slave ports. The details about the port mapping are given in the wrapper architecture. The figure below shows the wrapper implementation in the design.

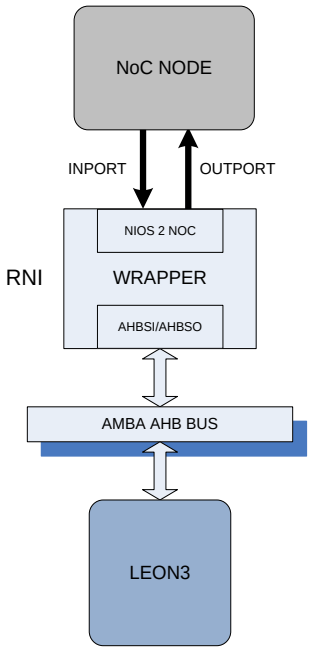


Figure 26: The NoC Wrapper

5.2.1. Wrapper Architecture

It has been described earlier that the wrapper has been build for the adaptation of the NoC to the design according to its AMBA AHB interface. The signals of the Avalon bus are assigned to its corresponding AMBA AHB signals and vice versa are shown in the table below.

Avalon	Direction	AMBA
Avalon_slave_read	Input	HWRITE
Avalon_slave_write	Input	
Avalon_slave_chipselect	Input	HSEL , HREADY
Avalon_slave_address	Input	HADDR
Avalon_slave_writedata	Input	HWDATA
Avalon_slave_readdata	Output	HRDATA

Table 5: RNI wrapper signals

The table shows that there is one to one correspondence between the two bus architectures. As can be seen the signals are directly allocated to their corresponding counterpart, except the two distinct signals these are the ‘read / write’ (Avalon_slave_read / Avalon_slave_write) signals [24]. As far as the AMBA AHB is concerned, it has a single write signal for both read and write operations, named HWRITE [2]. When it is ‘high’ it corresponds to a write operation and when ‘low’ it corresponds to the read operation. For the wrapper, the AMBA signal only has to generate two of Avalon signals; one will be the negation of the other.

Another signal is Avalon_slave_chipselect that need to be translated by using two AHB signals. These are HSEL and HREADY. When both are high they correspond to the Avalon_slave_chipselect. There are some more AHB signals also that are used in the RNI wrapper design, like, the HCONFIG signal is used to insert the IP in the system, the HREADY is used to indicate the transfer completion [2], [31].

5.3. RNI adaption in the design

The RNI is added into the GRLIB as AMBA AHB slave. Below is the procedure that is adapted to add RNI as an IP core in the design. To adapt an IP core with AMBA interfaces into GRLIB, the corresponding AMBA signals has to be declared as standard IEEE-1164 signals. Once done it is a matter of assigning these signals to the corresponding field of the AMBA record types declared in GRLIB, and to define the plug&play configuration information. The plug&play configuration utilizes the constants and functions declared in the GRLIB AMBA 'types' package, these are the HINDEX generic, HADDR and HMASK generics. The RNI is named kth_amba_noc_rni.vhd in the design and is added into leon3mp.vhd file as a component. Below is the section of the code for the RNI implementation [14]:

```
library ieee; use ieee.std_logic_1164.all;
library grlib; use grlib.amba.all;
USE WORK.NoC_wrapper_package.all;
USE WORK.NoC_parallel_package.all;

entity kth_amba_noc_rni is
generic(
       hindex : integer := 0;
       haddr : integer := 0;
       hmask : integer := 16#fff#;
       row_pos : integer := 0;
       col_pos : integer := 0;
       channel_width : integer := 9
       );
port(
       clk : in std_logic;
       reset : in std_logic;
       -- AHB slave ports
       ahbsi : in ahb_slv_in_type;
       ahbso : out ahb_slv_out_type;
       -- NoC Switch Ports
       inport:in outpacket;
       outport:out inpacket
       );
End;

architecture rtl of kth_amba_noc_rni is

-- component to be interfaced to GRLIB
……..
Begin

-- plug&play configuration
constant hconfig : ahb_config_type := (
0 => ahb_device_reg ( 0, 0, 0, 0, 0),
4 => ahb_membar(haddr, '1', '1', hmask), others => zero32);

……..

-- amba ahb records mapping
read_buffer:rni_memory
generic map (width => read_width)
port map(

……..

write_buffer:rni_memory
generic map (width => write_width)
port map(
```

```
……..

end rtl;
```

As shown above the entity is declared according to the standard required by the GRLIB for its IP core design. There is also a plug&play configuration, it can be seen that all the fields in ‘ahb_device_reg’ are marked with ‘0’, this is because the current IP is not in the device ID list of the GRLIB IP library, but even it is marked with zeros, it can be instantiated in the leon3mp design file. The dual port RAM is generated by the syncram_dp generic available in GRLIB. The details about this procedure can be seen in [11].

The configuration record from each AHB unit is sent to the AHB bus controller via the HCONFIG signal. For the RNI the HINDEX and ahbso are both numbered ‘7’, because in the slave record register the numbers from ‘0’ to ‘6’ are already occupied by the existing IPs. To add one more slave, the number ‘7’ is extracted from the list of unused slave modules. The bus controller creates the configuration table automatically, and creates a read-only memory area at the desired address (default 0xFFFFF000). It is also necessary to assign the HCONFIG to ‘ahbso.hconfig’ in the design to generate a True dual port RAM [14].

5.3.1.RNI memory

The RNI memory is modified to be able to use it in the current 2x2 NoC design. The RNI memory is created by port mapping the dual port RAM named ‘syncram_dp’ that is provided in the GRLIB. The IP can support different venders that are selectable from the configuration record; this scheme makes the RNI memory vender independent. There are two separate memories for reading and writing the messages. The syncram_dp is port mapped in the form of the ‘read_buffer’ and ‘write_buffer’ in the RNI design file. When the design is compiled in Quartus design environment, two dual port RAM memories are automatically generated, that can be seen in RAM summary in the next chapter.

5.4. AHBROM modification

In order to provide a mechanism so that the Leon3 NoC can generate the four separate PROMs at the time of booting, the AHBROM is to be modified. The 2x2 NoC design is provided with a separate ‘ahbrom.vhd’ file for each Leon3, they are distinguished by their names like ahbrom0, ahbrom1 so on. Then to add the software on the FPGA each AHBROM is equipped with a Memory Initialization File (MIF). It will provide the initial boot program to the Leon3 processor.

The processor status can be monitored through the Multiprocessor Status Register. The STATUS field in this register indicates if a processor is halted ‘1’ or running ‘0’. A halted processor can be reset and restarted by writing a ‘1’ to its status field. After reset, all processors except processor 0 are halted. When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits [15].

The modification to add .mif in done in two steps, first a romgen.vhd file is created in which the altsynram is instantiated. There is an ‘init_file’ generic that passes a string indicating the path of .mif file, present in the project. After that the romgen.vhd is instantiated in the ahbrom.vhd file. Here the modification is done by replacing the LUT by component romgen.

It is to be noted that the ahbrom and romgen file are available for each Leon3 in separate and indentified by their numerals.

The '.mif' format is shown below, the initials width and depth denote the size of memory and the address and data radix will show how the contents will be displayed in the content fields. Current format is set to hex.

```
-- Memory Initialization File (.mif)

WIDTH=32;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
  00000000 : 00004020;
  00000001 : 00001010;

  …… ;

  000000FE : 003DA03A;
  000000FF : 003DA03A;
END;
```

5.5. Top Design File

The GRLIB IP library supports the multiple Leon3 processor and or related IPs and peripherals on a single bus, such a system can be called single bus based system. If there are more buses in the design then it can be called as the multiple bus systems. The available design facility in the GRLIB is shown below [16]. As can be seen in the figure 27, several IPs and Leon3 processors are connected to the AMBA bus. It is the limitation of the Gaisler tools that it cannot support multiple bus systems.

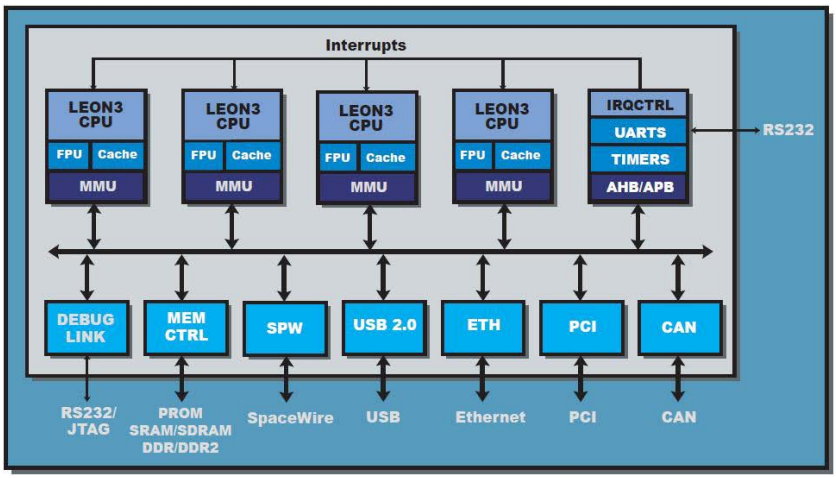


Figure 27: Multiple Leon3 processors on single AMBA bus

For the thesis there is a requirement to have multiple bus systems in a project that is, each system must be equipped with a Leon3 processor, an AMBA AHB bus and the required

peripherals like RNI, AHBROM, AHBRAM and JTAG port in separate. They can only interact with one another by using ports.

In order to deal with this requirement a top design module has been designed. The ‘top design module’ has four, Leon3 processor based bus systems that are instantiated along with the NoC design. The Leon3 processor and other required peripherals are separated by keeping the design files in separate folders and also by enumerating them with numbers from 0 to 3. As shown below, in the figure each box represents a complete Leon3 based system with its separate AHB bus.

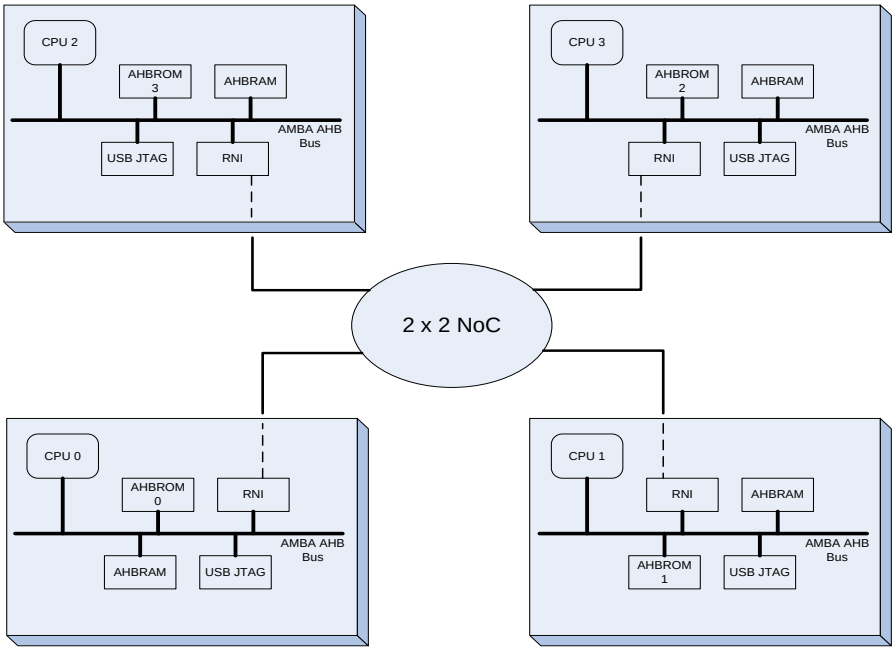


Figure 28: The Top module

The NoC design offers four nodes, i.e. from node0 to node3 to make a 2x2 NoC, in the design file kth_2x2_noc.vhd. An exploded view of the design is shown below shows how here each node of the NoC is connected to its corresponding Leon3 processor through the RNI.

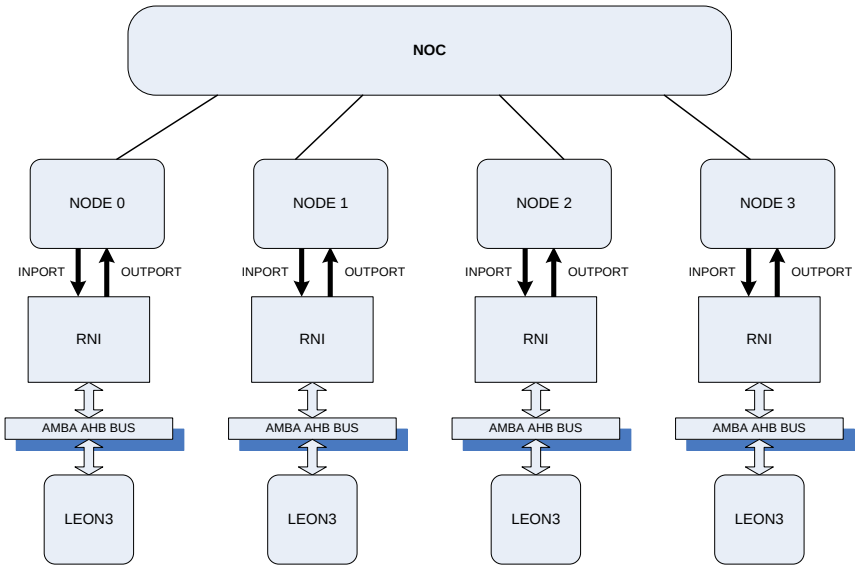


Figure 29: 2x2 NoC Design

The entity of the Top Module is shown below. Note that the entity has only the clock and reset ports available, that are used to provide the FPGA with the clock and reset. There are separate config file for each Leon3 processor in work library, denoted by conf_0 to conf_3. The values updated in the config.vhd file are copied to corresponding conf_X files when the design is configured from the GUI.

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.NoC_wrapper_package.all;
USE WORK.NoC_parallel_package.all;
library grlib;
use grlib.amba.all;
. . . . .
use work.conf_0.all;
use work.conf_1.all;
use work.conf_2.all;
use work.conf_3.all;

---- ENTITY ----

entity top_leon3_sys is
port(
	reset : in std_ulogic;
	clk : in std_ulogic
);
end;
```

In the Top design module the leon3mp design is instantiated four times with components leon3mp0 to leon3mp3. Each leon3mp instantiates an RNI memory, and separate ahbrom and romgen components.

For the instantiation of NoC design for the four nodes the inport and outport of the design are port mapped in the kth_noc_2x2.vhd file. The NoC design component in top design file is shown below:

```
component kth_noc_2x2 is
port(
	reset : in std_ulogic;
	clk : in std_ulogic;
	inport_0 :out outpacket;
	outport_0 :in inpacket;
	inport_1 :out outpacket;
	outport_1 :in inpacket;
	inport_2 :out outpacket;
	outport_2 :in inpacket;
	inport_3 :out outpacket;
	outport_3 :in inpacket
);
end component;
```

As shown, two ports namely the inport and outport are used for transferring data packet to or from each node respectively.

5.5.1.Design Synthesis

The top level design is compiled by Altera's Quartus II development environment. The design is added as a project along with all required packages and related design files. Once the compilation is done successfully, the peripherals added in the design can be seen by the RTL diagram and by the compilation report. The whole design is internal to the FPGA. The clk and reset ports can be seen in the figure below.

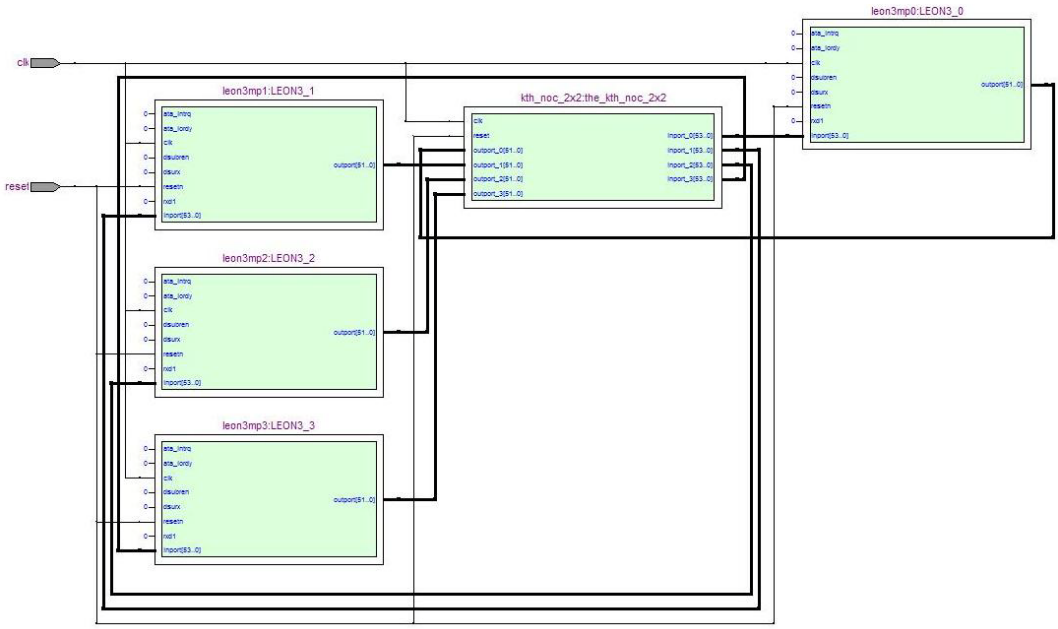


Figure 30: RTL view of the 2x2 Leon3 based NoC system

Each Leon3mp module is connected to the NoC through two ports one for data input and other for data output.

5.6. Graphical User Interface

In order to provide a user friendly environment to generate a new Leon3 system and to configure it according to user's need a Graphical User Interface (GUI) has been developed in Microsoft Visual Basic 2008 environment. The GUI is provided with all the features required to compile and synthesize the new system and to edit an existing one. The following sections will explain different sections of the GUI.

5.6.1. Overview

The GUI is created by Visual Basic's 'windows forms application' project file. That is a project for creating an application with a Windows user interface (.NET Framework 3.5). In total GUI has five forms, one is the main GUI form, and others are the node forms, for the configurations of four nodes. There are separate buttons to configure each node. Also the buttons for the generation of a new system, for the loading of an existing system, and for the simulation and synthesis by using Modelsim and Quartus design environments, respectively. There is a 'file/folder name' text field to display the path of the design project, and also a label that displays the status of the running activity.

For the new system generation the template design has to be copied first to the computer, the default location is c drive. The computer must have installed the Windows, Microsoft Visual Basic 2008, the GRTools and Cygwin with 'wish' and tcl/tk 8.4 or 8.5. If there is a requirement to save the design to some other location the path must be edited in VB design file. The design must be copied to the 'designs' folder of the GRLIB IP library. If the design is copied to some low hierarchy level then the hierarchy level must be set first in the design's 'Makefile'.

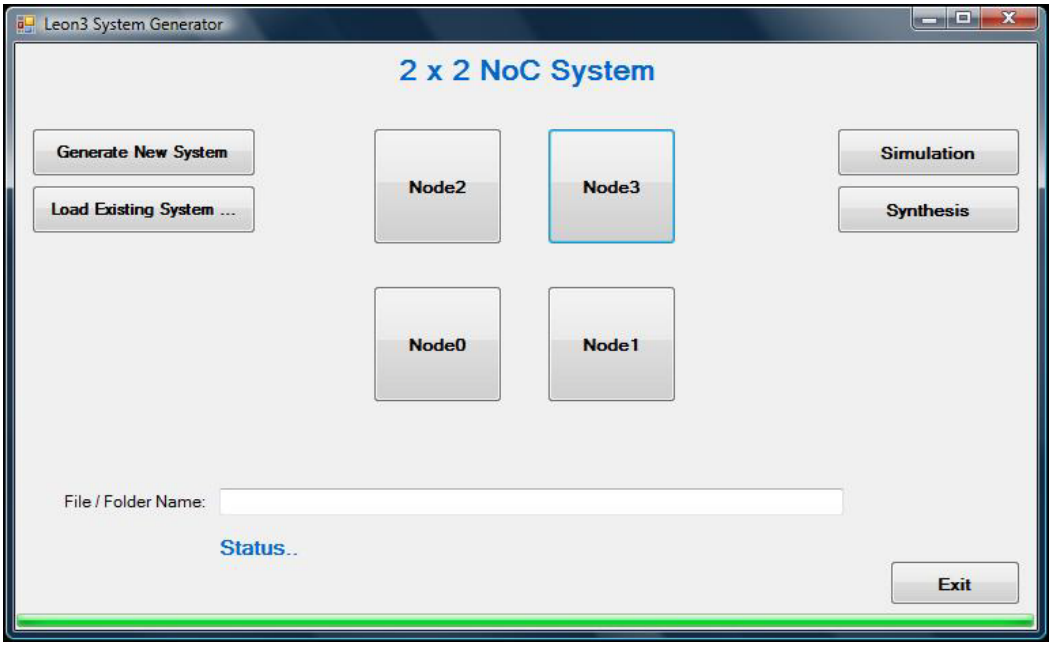


Figure 31: GUI for 2x2 NoC system

5.6.2. Generation of a new system

To generate a new Leon3 system there is button available on the GUI with label 'Generate New System'. For the generation of the new system, first the user will be asked about the location where to build the new system. The location is browsed by using windows folder browser scheme in VB environment. It can be done either by making a new folder or by an existing folder location. The user can interrupt this process and come out of it any time. After finalizing the required location the design will be copied to that place with all the required files and folders. While finalizing the location, the user has to confirm it again otherwise the previously saved folder will be replaced by the new one.

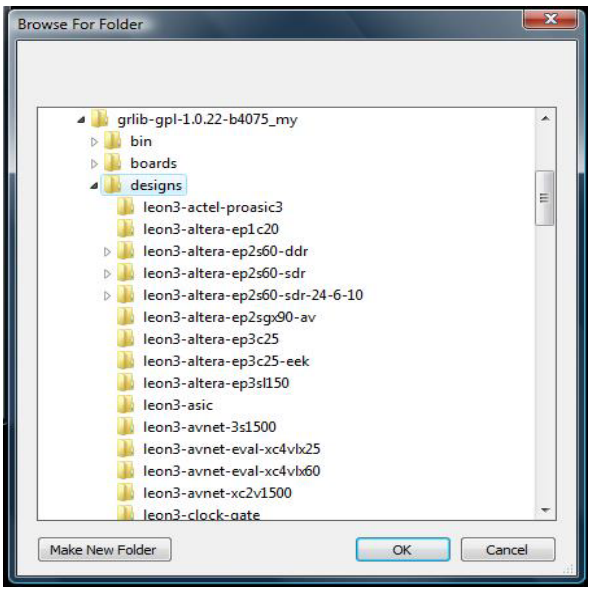


Figure 32: Folder browser for generating new system

5.6.3. Open an existing system

If the user requires to open or to edit an existing system, for that purpose there is a button available on the GUI with label ‘Open existing system …’. This will open a file browser dialog window, using VB ‘.NET’ command. In order to ease the project selection there is provided a filter which will find the ‘Makefile’ in a specific folder, as shown below. This is useful to quickly confirm that the user is browsing in right folder or not. If no Makefile exist there then, there is a possibility that either no design exists or there is a requirement for generating a new system. The open button in this dialog will load the existing design. The project path will also be displayed in the GUI text filed.

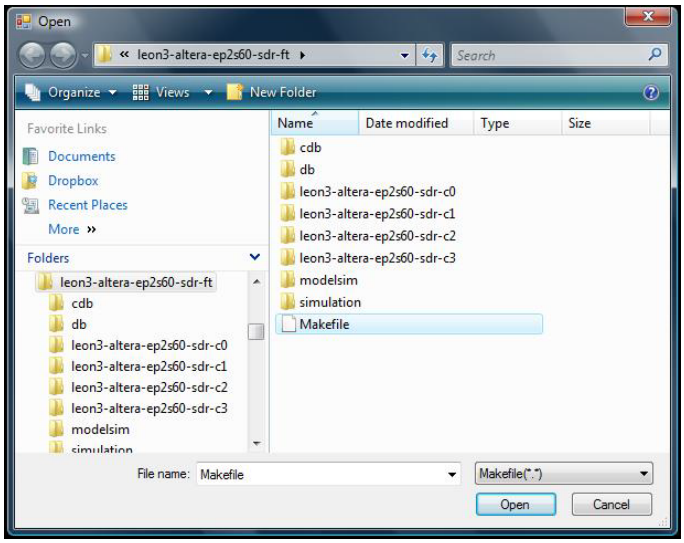


Figure 33: Opens an existing system

5.6.4. Configuring Nodes

The GUI is provided with four buttons namely ‘Node 0’, ‘Node 1’, ‘Node 2’ and ‘Node 3’ for the Leon3 configuration. The Node buttons opens a separate form for each node. The figure below shows the form for Node0 only, rest of the forms is same like this one. Each form has a ‘Configure’ button and a ‘Done’ button The configure button is a used to open the Leon3 Configuration GUI, here the user can make all the necessary changes in the configuration, that are saved on the config.vhd file. For the 2x2 NoC each Leon3mp is provided a separate ‘config.vhd’ file, they are named, conf_0.vhd for Node 0, conf_1.vhd for Node1 and so on to differentiate them.

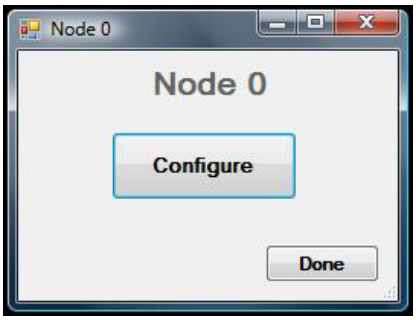


Figure 34: Node configurations

The ‘Done’ button is provided to complete the Node configuration procedure. When ‘Done’ button is clicked the data from the config.vhd file is copied to the corresponding conf_X.vhd file so as to save the parameters for that particular Leon3 design file.

5.6.5.Simulation

After the nodes configuration the user can simulate the system with the help of ‘Simulation’ button on the GUI. The simulation of the system is completed by using Mentor graphics Modelsim environment, as shown.

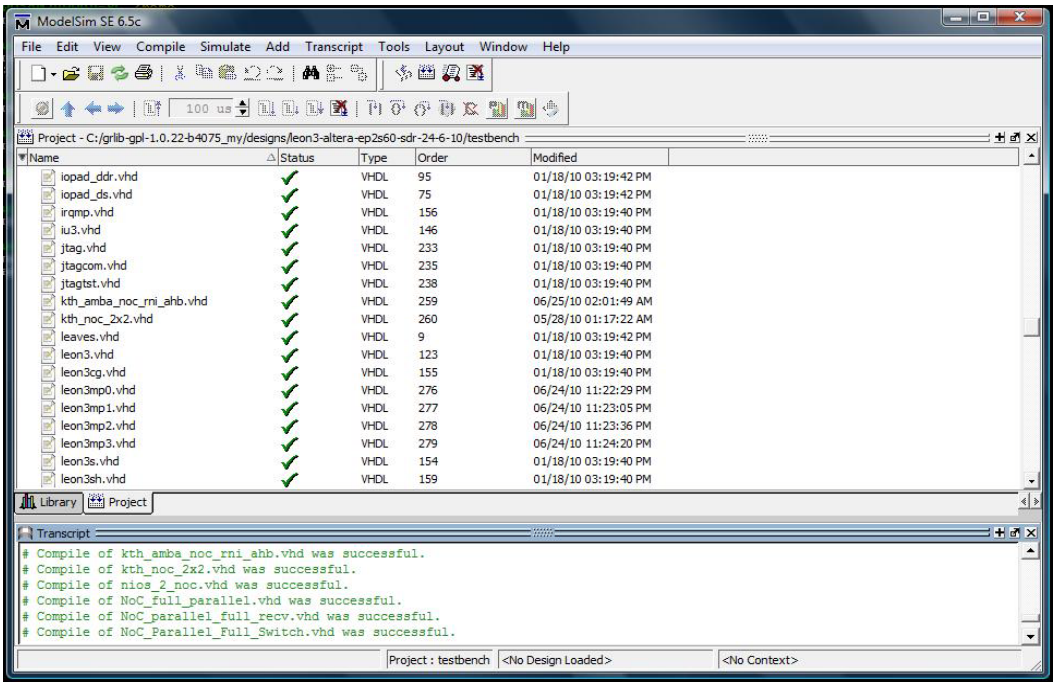

ModelSim SE 6.5c
File Edit View Compile Simulate Add Transcript Tools Layout Window Help
Project - C:/grlib-gpl-1.0.22-b4075_my/designs/leon3-altera-ep2s60-sdr-24-6-10/testbench
Name Status Type Order Modified
iopad_ddr.vhd VHDL 95 01/18/10 03:19:42 PM
iopad_ds.vhd VHDL 75 01/18/10 03:19:42 PM
irqmp.vhd VHDL 156 01/18/10 03:19:40 PM
iu3.vhd VHDL 146 01/18/10 03:19:40 PM
jtag.vhd VHDL 233 01/18/10 03:19:40 PM
jtagcom.vhd VHDL 235 01/18/10 03:19:40 PM
jtagtst.vhd VHDL 238 01/18/10 03:19:40 PM
kth_amba_noc_rni_ahb.vhd VHDL 259 06/25/10 02:01:49 AM
kth_noc_2x2.vhd VHDL 260 05/28/10 01:17:22 AM
leaves.vhd VHDL 9 01/18/10 03:19:42 PM
leon3.vhd VHDL 123 01/18/10 03:19:40 PM
leon3cg.vhd VHDL 155 01/18/10 03:19:40 PM
leon3mp0.vhd VHDL 276 06/24/10 11:22:29 PM
leon3mp1.vhd VHDL 277 06/24/10 11:23:05 PM
leon3mp2.vhd VHDL 278 06/24/10 11:23:36 PM
leon3mp3.vhd VHDL 279 06/24/10 11:24:20 PM
leon3s.vhd VHDL 154 01/18/10 03:19:40 PM
leon3sh.vhd VHDL 159 01/18/10 03:19:40 PM
Library Project
Transcript
Compile of kth_amba_noc_rni_ahb.vhd was successful.
Compile of kth_noc_2x2.vhd was successful.
Compile of nios_2_noc.vhd was successful.
Compile of NoC_full_parallel.vhd was successful.
Compile of NoC_parallel_full_recv.vhd was successful.
Compile of NoC_Parallel_Full_Switch.vhd was successful.
Project : testbench
<No Design Loaded>
<No Context>

Figure 35: Modelsim simulator

For the 2x2 NoC design the simulation software will be started with all the required files already added in the project. The compiled simulation model is in the ./modelsim sub directory. There is a modelsim.ini file that contains the necessary VHDL library mapping definitions. The GUI activates the GRLIB command ‘`make vsim-launch`’, by using VB shell function. When doing, the design is compiled and simulated by generating Modelsim project file.

The script for the project file is already generated for the user and saved in the template design folder for 2x2 NoC. Once the design is opened the user can make all necessary changes and build the design again and start the simulation. The procedure for for compilation and simulation with the Modelsim design environment can be found in Modelsim User’s Manual.

5.6.6.Synthesis

There is a ‘Synthesis’ button available on GUI that gives the user a provision to compile and synthesize the whole Leon3 based 2x2 NoC system in Altera’s Quartus II design environment. For starting the design synthesis the program must first be loaded into the GUI.

This can be achieved by either generating the new system or by opening an existing system, as discussed in previous sections. Once the Synthesize button is pressed the Quartus II design environment will be launched with all necessary files for the system, which can be seen in the files menu of the Quartus II project navigator, as shown in figure below.

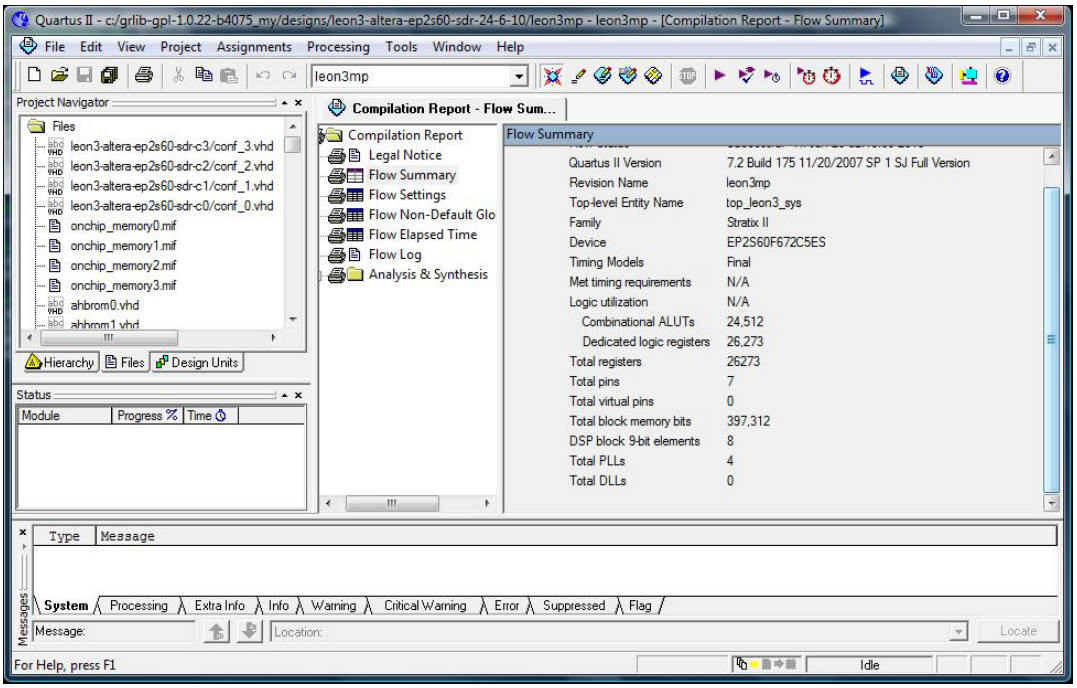


Figure 36: Quartus II design environment

Similar to the case of simulation as discussed earlier the scripts has been already saved in the template design of 2x2 NoC for the users by using '`make scripts`' command. The command had generated two project files for Quartus, one for an EDIF flow where a netlist was created with synplify and one for a Quartus-only flow [14]. The project files are named top_leon3_sys.qpf and top_leon3_sys_synplify.qpf. The interactive synthesis and place and route operation is achieved through the command '`make quartus-launch`' that is activated in the GUI by using Visual Basic shell function. The Quartus can also be started manually with '`.qpf`' project file from the template designs folder of 2x2 NoC design [14].

Chapter 6

6. Evaluation

This chapter will highlight the testing and evaluation procedure of the 2x2 NoC design. The chapter also describes how the RNI memory is adapted in the design, the simulation of the 2x2 NoC design to verify its functionality and the synthesis of the design in Quartus II environment. The chapter also describe the procedure to port the design on a FPGA board.

6.1. Adaption of RNI memory in leon3mp project

As discussed earlier, that in GRLIB IP library the entities can be added as AHB slaves if they are written with proper format of AHB Slave record type [14]. In order to verify whether the RAM has been added in the design or not, first the design has to be compiled in Quartus II environment. Then, after the successful compilation of the design, 'RAM summary' can be seen in the compilation report. It is evaluated that two dual port RAMs are automatically generated by using altera_syncram_dp design, from the Altera's mega-wizard function. The details if the RNI memory is shown below.

RAM Summary

```
leon3mp0:LEON3_0|kth_amba_noc_rni_ahb:RNI|my_rni_memory:read_buffer|syncram_
dp:
the_altsyncram|altera_syncram_dp:\alt:x0|altsyncram:u0|altsyncram_j7r:
auto_generated|ALTSYNCRAM

leon3mp0:LEON3_0|kth_amba_noc_rni_ahb:RNI|my_rni_memory:write_buffer|syncram
_dp:
the_altsyncram|altera_syncram_dp:\alt:x0|altsyncram:u0|altsyncram_l7r:
auto_generated|ALTSYNCRAM
```

The RAM summary shows that the generated RAMs are of type syncram_dp, the RNI memory is represented by the two dual port RAMs named the read buffer and write buffer. The information for generating the RNI memory is extracted from the my_rni_memory.vhd file. Note that address space of 'read_buffer' is 10 bits and it generates a RAM is of size 1KB and the address space of the 'write_buffer' is 11 bits and the generated RAM is of size 2KB. For the Dual port RAMs the total size of read_buffer is 2KB and that of write_buffer is 4KB, that creates the RNI memory of size 6KB for each Leon3 processor.

6.2. Modification of AHBROM

The AHBROM is instantiated inside the leon3mp.vhd file; it is implemented as a 32bit On-Chip ROM used to generate the boot application program. The AHBROM is modified to accommodate the (Memory Initialization File) MIF, to provide an initial boot program to the resource (Leon3 processor). The MIF file is provided in separate for each AHBROM for each Leon3 processor. The evaluated on-chip ROM can also be seen in the RAM summary after the compilation of the design in the Quartus II environment, as shown below.

RAM Summary

```
leon3mp0:LEON3_0|ahbrom0:\bpromgen:brom|ROMGEN0:romgen_inst|altsyncram:
altsyncram_component_0|altsyncram_mt71:auto_generated|ALTSYNCRAM
```

6.2.1. Generating Boot Image

The Gaisler also provides a utility program that can create the boot images of the program that are compiled with the BCC the Leon3 cross compiler. The program is first compiled in the BCC environment, then the register files of the Integer Unit of Leon3 are initialized, then the control, wait states and memory configuration registers are set. The application is installed into the RAM and by setting the stack pointer on the top of the RAM the application started [13].

For the Leon3 processor the –memc <addr> set the address of the memory controller register, the –gpt <addr> sets the address of the timer unit control register and –uart <addr> will set the address of uart control registers [13].

6.3. Testbench implementation

The simulation of 2x2 NoC design using testbench verifies the working of the complete system and particularly the working of RNI in the design. The structure and working of the testbench is described below.

6.3.1. Structure

The structure of the testbench is divided into small sections. The testbench has declared the RNI and NoC designs by adding the components from the files kth_amba_noc_rni.vhd and noc_full_parallel.vhd respectively. There are also print functions that allow the outputs and the assertions to appear on Modelsim console and waveform window, these are named as, ‘print_std_logic’, ‘print_vector’ and ‘print_word’. In addition there are clock and reset signals and RNI port mapping.

6.3.2. Working

In the testbench the ‘process’ statement allows a test pattern in the network design that allows the read and write operations on the NoC and vice versa. This is done in following manner. There are number of procedures in the testbench, they are for initialization of NoC memory, the Leon3 to NoC read/write and the idle cycle etc. There is also a process for reading and writing to the RNI status registers. When Leon3 wants to read the data sent by the NoC, then it first has to read the interrupt register and find out which source has sent the data, then it first reads the message length and afterwards start receiving the data. When Leon3 wants to write, then it will first write it to a command register, on the basis of this command register, it is then decided where to send the data.

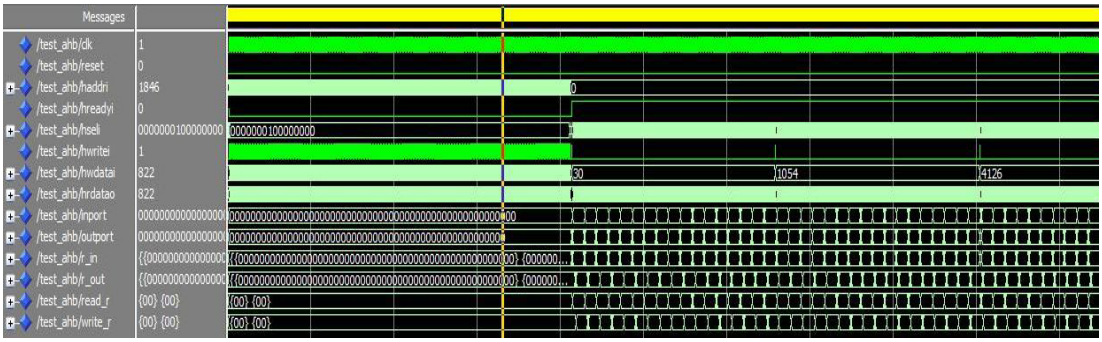


Figure 37: The testbench simulation

The FSM that receive data from the processor to the NoC is wrapped around by AHB bus wrapper. It first sends out the setup packet in order to read the input. Then it waits till all the data is sent. Afterwards that it starts reading the memory. When all data is read then it goes back to idle state. Above is shown the waveform of the simulation, it first send 15 words to each resource R0, R1, R2 and R3, and then check the data from R3. The testbench has instantiated R3 only, so first R3 send messages to all RNIs and then check if the right message appears on the correct output by reading the RNI. In the testbench any resource can be replaced by R3 for the testing purposes.

6.4. Synthesis using Quartus

The synthesis and place&route of the complete 2x2 NoC design is done in Altera's Quartus II design environment. In the top design module, The 4 Leon3 processor systems are connected to the NoC, as explained earlier. After the successful compilation of the top design module, the results can be seen on the compilation report the summary, tabulated below.

S.No.	Status	Results
1.	Family	Stratix II
2.	Device	EP2S60F672C5ES
3.	Logic Utilization	76 %
4.	Combinational ALUTs	24.508 / 48.352 (51 %)
5.	Dedicated Logic Register	26.133 / 48.352 (54 %)
6.	Total Registers	26133
7.	Total Pins	7 / 493 (1 %)
8.	Total Block Memory Bits	528,384 / 2.544.192 (21 %)
9.	Total PLLs	4 / 6 (67 %)
10.	Total DLLs	0 / 2 (0 %)

Table 6: Compilation report summary

It is evident from the table above that, in the current design the logic utilization is 76 %, which is quite high as compared to the previous NoC design. This may be because Leon3 is using more flip-flops than Nios II on the same FPGA. This is shown by total memory block usage is 21% comparing to Nios II. From the above results it is seen that more modules can also be added into the design, like timer unit, debug support unit etc. This design can be regarded as the simplest possible design for checking the functionality of the 2x2 NoC. The NoC utilizes only 4KB of scratch pad memory and 6 KB RNI memory for each Leon3 processor. The system can be enhanced if more logic is enabled from its configuration.

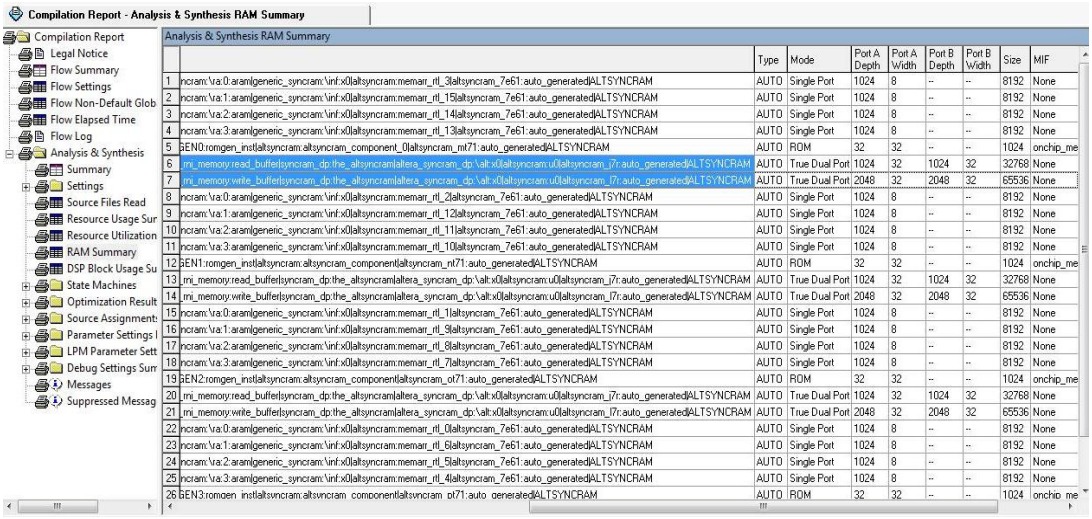


Figure 38: RNI evaluated in Leon3mp design

The evaluated AHBROM, AHBRAM and RNI Memory are given in the table below, the 4KB of AHBRAM are shown as 4 blocks of 1KB, the detail can be seen in [14]. The AHBROM has a .mif file for each Leon3 processor. The RNI memory is added as two True Dual Port RAMs, 2KB for read_buffer and 4KB for write_buffer.

Name	Mode	Part A Depth	Port A Width	Port B Depth	Port B Width	MIF
ahbram0	Single Port	1024	8	--	--	None
ahbram0	Single Port	1024	8	--	--	None
ahbram0	Single Port	1024	8	--	--	None
ahbram0	Single Port	1024	8	--	--	None
ahbrom0	ROM	32	32	--	--	onchip_memory0.mif
rni_memory: read_buffer	True Dual Port	1024	32	1024	32	None
rni_memory: write_buffer	True Dual Port	2048	32	2048	32	None
ahbram1	Single Port	1024	8	--	--	None
ahbram1	Single Port	1024	8	--	--	None
ahbram1	Single Port	1024	8	--	--	None
ahbram1	Single Port	1024	8	--	--	None
ahbrom1	ROM	32	32	--	--	onchip_memory1.mif
rni_memory: read_buffer	True Dual Port	1024	32	1024	32	None
rni_memory: write_buffer	True Dual Port	2048	32	2048	32	None
ahbram2	Single Port	1024	8	--	--	None
ahbram2	Single Port	1024	8	--	--	None
ahbram2	Single Port	1024	8	--	--	None
ahbram2	Single Port	1024	8	--	--	None
ahbrom2	ROM	32	32	--	--	onchip_memory2.mif
rni_memory: read_buffer	True Dual Port	1024	32	1024	32	None
rni_memory: write_buffer	True Dual Port	2048	32	2048	32	None
ahbram3	Single Port	1024	8	--	--	None
ahbram3	Single Port	1024	8	--	--	None
ahbram3	Single Port	1024	8	--	--	None
ahbram3	Single Port	1024	8	--	--	None

ahbrom3	ROM	32	32	--	--	onchip_memory3.mif
rni_memory: read_buffer	True Dual Port	1024	32	1024	32	None
rni_memory: write_buffer	True Dual Port	2048	32	2048	32	None

Table 7: RAM Summary

6.5. Design compilation

The compilation of the application program can be done by using the BCC cross compiler for Leon3. Compilation takes place as shown in the following flow graph.

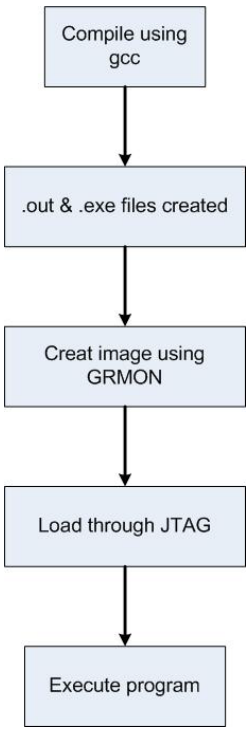


Figure 39: The normal execution flow graph

The program is first added into the project folder; that include the c-source code and all the required header files. Once the program is added into the project; following command is used to compile the c-code.

```
sparc-elf-gcc LED_test.c –o LED_test.exe
```

The above command will generate an '.exe' file. Then the GRMON debugger is started by using command.

```
grmon-eval –altjtag -u
```

The GRMON will then started and establish a JTAG connection and display the list of the all the attached peripherals, as shown in chapter 4. Then the GRMON is ready to load the program, the program is loaded using the command.

```
grmon> load LED_test.exe
```

When the program is loaded it creates a boot image as shown:

```
section: .text at 0x40000000, size 54368 bytes
section: .data at 0x4000d460, size 2064 bytes
section: .jcr at 0x40024e68, size 4 bytes
total size: 56436 bytes (90.9 kbit/s)
read 196 symbols
entry point: 0x40000000
```

Then the execution is done by using simple run command, as shown

```
grmon> run
```

However the above whole scheme is for booting only one Leon3 processor, in order to boot a complete NoC system there is a need to define some mechanism. One way of doing this is to create a subroutine with which the boot image can be converted to '.mif' file. As shown in the following flow graph:

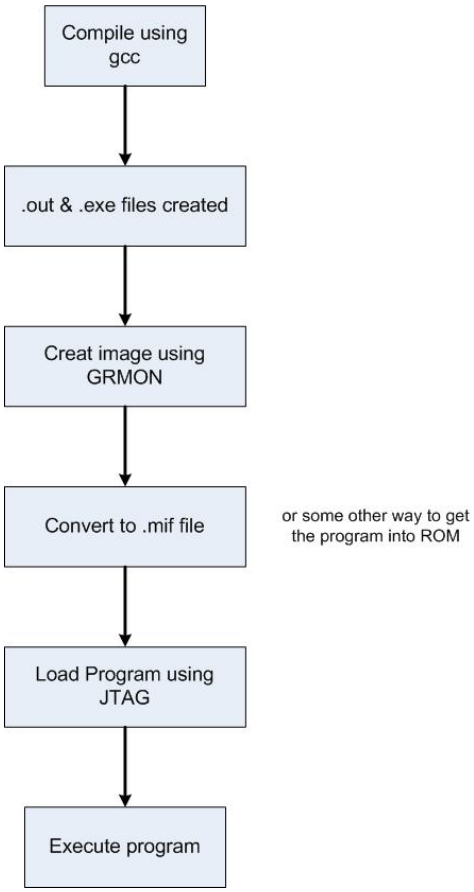

Compile using gcc
.out & .exe files created
Creat image using GRMON
Convert to .mif file
or some other way to get the program into ROM
Load Program using JTAG
Execute program

Figure 40: the flow graph with .mif converter

If the image file could be converted to '.mif' then it could be possible to run the complete the Leon3 NoC system. That task is put as the future possibility. When that part will be done, then the GUI can be modified for the each Node as shown in the figure below. As can be seen there are two buttons, one for the configuration of the Hardware for a node and other for the compilation of the loaded software.

Figure 41: Node configuration and compilation

6.6. Design porting on FPGA board

In order to debug and to see the working of the Leon3 based NoC. The design has to be tested on the Nios development board. To test the design on the hardware platform there is a requirement of the device driver and memory map of the system. The device driver and the 'system.h' file of the former 4x4 NoC design has to be used for this purpose.

In the device driver file the base address is to be changed to the current RNI_BASE_ADDRESS and LED_PIO_ADDRESS are to be defined. The software for the design will check the successful transfer of the packets from one node to the other and their status will be displayed on separate LEDs on the development board. The software porting on the hardware will be done using GRMON debug monitor.

6.6.1. Accessing Hardware

The Stratix II FPGA based Nios development board is the required hardware to test the design. To access the hardware, the GRGPIO port is to be used. It can be used to connect the Leon3 based system to the LEDs on the board. The LEDs are controlled by using the GRGPIO core's output and direction registers. The Altera's macro and functions like, IOWR_ALTERA_AVALON_PIO_DATA used in the system.h file are to be replaced, with the pio function provided by Gaisler [8].

The mapping of the system is available in the '.qsf' file, in the design's main folder. There the base address of the core is placed where the LED is connected. The pseudo code to control the LEDs is given below [8]:

```
volatile unsigned int *pio = (int *) 0x<base address of core>;
pio[2] |= (1 << GpioLine); /* Set GpioLine as output */
pio[1] |= (1 << GpioLine); /* Drive GpioLine high */
pio[1] &= ~(1 << GpioLine); /* Set GpioLine low */
```

The program will start from the memory area where the core registers are mapped, in 'leon3mp.vhd' file. For the leon3-altera-eps60-sdr design it can be seen on the output as:

```
# apbctrl: slv5: Gaisler Research General Purpose I/O port
# apbctrl: I/O ports at 0x80000500, size 256 byte
```

Here the output and direction registers are affected by the bit X, the assignment is shown below:

```
pio[2] |= (1<< X); /* Set GpioLine as output */
```

It will set bit X in the register pointed at pio[2] to 1 and leave all the other bits in pio[2] unchanged. Whereas the assignment below will set bit X to zero and leave all the other bits unchanged.

```
pio[2] &= ~(1<< X);
```

To know which line on the GPIO port to drive in order to control a LED, the mapping in boards/altera-ep2s60-sdr/leon3mp.qsf is shown below:

```
set_location_assignment PIN_AD25 -to gpio[1] #out_port_from_the_led_pio[1]
set_location_assignment PIN_AC25 -to gpio[2] #out_port_from_the_led_pio[2]
set_location_assignment PIN_AC24 -to gpio[3] #out_port_from_the_led_pio[3]
set_location_assignment PIN_AB24 -to dsuact #out_port_from_the_led_pio[4]
set_location_assignment PIN_AB23 -to gpio[5] #out_port_from_the_led_pio[5]
set_location_assignment PIN_AB26 -to gpio[6] #out_port_from_the_led_pio[6]
set_location_assignment PIN_AB25 -to gpio[7] #out_port_from_the_led_pio[7]
```

If the required LED is not mapped to the GPIO port then ‘leon3mp.qsf’ must be modified to the required LED.

For example to drive out_port_from_the_led_pio[1] high (and leave all other lines as inputs) the code section is shown below,

```
volatile unsigned int *pio = (int *) 0x80000500;
pio[1] = 0x2; /* or (1 << 1) */
pio[2] = 0x2;
```

Here the pio[x] will access the register at byte offset 4*x on the GRGPIO core.

6.7. Changes required in GRLIB

There is a bug in the GRLIB core design file leon3mp.vhd supplied by the Gaisler; when downloaded from their website. In this file the apbo are assigned the wrong value [8], shown in the code section below. Here the apbo(7) should be replaced by apbo(4), as pindex is also 4, it must be corrected to run the simulation correctly.

```
dcomgen : if CFG_AHB_UART = 1 generate
 dcom0 : ahbuart -- Debug UART
   generic map (hindex => NCPU, pindex => 4, paddr => 7)
   port map (rstn, clkm, dui, duo, apbi, apbo(4), ahbmi, ahbmo(NCPU));
 dsurx_pad : inpad generic map (tech => padtech) port map (dsurx, dui.rxd);
 dsutx_pad : outpad generic map (tech => padtech) port map (dsutx, duo.txd);
end generate;
nouah : if CFG_AHB_UART = 0 generate apbo(7) <= apb_none; end generate;
```

There is also a change required in the Leon3 configuration. The template design has the address value set to A00 by default for on-chip RAM, In order to enable the on-chip RAM only while disabling the SDRAM, the address must be changed to 400. The address of RNI memory is set to A00 in the design [8].

Chapter 7

7. Conclusion & Future work

This chapter will summarizes the results obtained at the end of thesis work and also discuss about the future work possibilities.

7.1. Conclusion

The thesis work is based on an existing 4x4 NoC system. It is found from the simulation that the Nios II processor based NoC design can be replaced by the Leon3 soft-core processor based design. The design was previously based on Avalon bus system that has been replaced by the AMBA AHB bus architecture, using wrapper concept. To simplify the design the 4x4 NoC design is reduced to a 2x2 NoC design by selecting four nodes in the package file and while keeping the packet size to 52 bits, so that design can be fit to a single FPGA board. The RNI is added as AHB slave memory that is added in the design by using the GRLIB plug&play feature. The RNI memory is modified by using syncram_dp design provided by Gaisler. The AHBROM is modified successfully and an initialization file is added in the design. The top design module is also designed successfully, resulting in a 2D Mesh 2x2 NoC system.

The GUI is successfully developed and tested for generating a complete Leon3 2x2 NoC system. The design is tested on different template design in the design folder of GRLIB IP library, to evaluate its functionality. The GUI is a single platform that provides solution for generating the Leon3 based 2x2 NoC design.

The simulation result shows that the NoC is able to work with AMBA AHB bus architecture. The design is also compiled and synthesized successfully. The synthesis result shows that the design is fitted successfully on the FPGA board. The procedure has been developed for debugging NoC using Leon3 on FPGA board. However to test the design using the device drivers by porting the design on the hardware need more time.

7.2. Future work

This thesis work opens various interesting research topics, for improving the design functionality and performance.

For the debugging of Leon3 based NoC system, the built-in functions of Nios II are to be replaced by the one provided by GRLIB, the task is yet to be completed. So far only the LEDs are accessed using the GRGPIO functions explained in evaluation chapter. The design is to be tested by replacing all macros and functions of the Altera's design by the one provided by Gaisler. The debug design should work with the pattern that every node has to transfer packet to other node, so the packet travel in anticlockwise direction in the network, and its status should be displayed on the LEDs.

So far the design is limited to compiling only one processor at a time, there has to be some mechanism that can start whole 2x2 NoC system. One possible way to do this is the

mentioned in chapter 6. Another possibility is, to make a sub-routine program that can form a layer between the GRMON debugger and NoC. That can work in such a manner that the command issue form GRMON will be passed to that sub-routine and then that sub-routine can convert it to '.mif' file. In this way the procedure for using GRMON for loading software will remain the same.

The design can be extended to 4x4 NoC design. For that purpose either similar boards can used to be connected with one another or the design can be shifted to some other bigger FPGA device. Extending the design can also result in problems like the bus contentions and deadlock that will need more complex logic and algorithms.

A debug support unit can also be added for the monitoring the status of system. The performance of the NoC design can also be evaluated by adding the timer module in the design. The Timer will monitor the time that a packet required to reach from source to destination.

The design can also be implemented on the other vender's template designs in GRLIB IP library. The fault tolerant version of the Leon3 can also be utilized to handle performance critical applications

The Complex applications like the Game of life or some high quality video streaming can also be tested on the NoC to see how the system actually work in these real life applications.

References

[1] http://intranet.cs.man.ac.uk/apt/APT_Research.php

[2] ARM Limited, *"AMBA Specification (Rev 2.0)"*, 1999, ARM IHI 0011.

[3] W. Dally and B. Towles, *"Principles and Practices of Interconnection Networks"*, Morgan Kaufmann Publishers, 2004. ISBN: 0-12-200751-4.

[4] Johnny Öberg, Roger Mellander, and Said Zahrai, *"The ABB NoC – a Deflective Routing 2x2 Mesh NoC targeted for Xilinx FPGAs"*, In the proceedings of the FPGA world Conference, September 2008.

[5] L. Benini and G. D. Micheli, *"Networks on chips: A new soc paradigm"*, In IEEE Computer, volume 35, pages 70_80, January 2002.

[6] Jiri Gaisler, *"GRMON Users Manual"*, AEROFLEX GAISLER AB, version 1.1.45, March 2009

[7] *"Nios Development Board, Reference Manual, Startix II Edition"*, Altera Corporation July 2005.

[8] http://tech.groups.yahoo.com/group/leon_sparc/

[9] Huynh Viet Thang, Pham Ngoc Nam, *"Prototyping of a Network-on-Chip on Spartan 3E FPGA"*, The University of Danang, Hanoi University of Technology, Vietnam,

[10] Juha-Pekka Soininen1, Axel Jantsch2, Martti Forsell1, Antti Pelkonen1, Jari Kreku1, and Shashi Kumar, *"Extending Platform-Based Design to Network on Chip Systems"*, VTT Electronics, Oulu, FINLAND, Royal Institute of Technology, Stockholm, SWEDEN

[11] Jiri Gaisler, Marko Isomäki, *"LEON3 GR-XC3S-1500 Template Design"*, Based on GRLIB, October 2006

[12] Wajid Hassan Minhass, Johnny Öberg, Ingo Sander, *"Design and Implementation of a Plesiochronous Multi-Core 4x4 Network-on-Chip FPGA Platform with MPI HAL Support"*, FPGA World'09, September 10, 2009, Kista, Stockholm, Sweden.

[13] Jiri Gaisler, *"Sparc-elf-mkprom User's Manual v1.0.12"*, Copyright 2004 Gaisler Research.

[14] Jiri Gaisler, Sandi Habinc, *"GRLIB IP Library User's Manual Version 1.0.22"* Copyright Aeroflex Gaisler, 2010

[15] *"GRLIB IP Core User's Manual Version 1.0.22"*, January 2010 Copyright Aeroflex"

[16] *"SPARC V8 32-bit Processor LEON3 / LEON3-FT Companion Core Data Sheet"*, March 2010, Version 1.1, Copyright Aeroflex Gaisler AB

[17] http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53

[18] Gustav Kynnefjäll, *"Performance tests of an open source multiprocessor system"*, TRITA-ICT-EX-2009:164, KTH, Sweden

[19] SPARC International, Inc. *"The SPARC Architecture Manual"*, Version 8th edition, 2008

[20] Fredrik Buddee, *"A Study of Open Source Soft Processor Cores"*, Master Thesis KTH, Sweden 2006

[21] Richard Zurawski, *"Embedded Systems Handbook (Industrial Information Technology)"*, ISBN 0-8493-2824-1, 2006

[22] http://www.ict.kth.se/courses/IL2207/1001/Lectures/L06-NoCIntro-Topology-Jan2010.pdf

[23] Davide Calusini, *"Implementation of an AMBA-NoC Interface and Performance Evaluation of a NoC-emulated AHB Bus"*, Master Thesis, ICT/ECS-2008-104

[24] *"Avalon Interface specification"*, Altera, April 2009

[25] Lutz Buttelmann, *"How to setup LEON3 VHDL simulation with Modelsim"*, Version 0.1 18.10.2007

[26] Jiri Gaisler, *"BCC - Bare-C Cross-Compiler User's Manual"*, Version 1.0.34, June 2010

[27] Jonas Floden, *"LEON Integrated Development Environment for Eclipse"*, Version 1.4.7 July 2010

[28] Michael Halvorson, *"Microsoft Visual Basic 2008, Step by Step"*, 2008

[29] Julia Case Bradley, Anita C. Millspaugh, "Programming in Visual Basic 2008", ISBN 9780073517209

[30] http://msdn.microsoft.com/en-us/library/xe736fyk%28VS.71%29.aspx

[31] *'Avalon Interface specification'*, Altera, April 2009

[32] http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=272&Itemid=31